Ю.Э. Корчагин

ПРОГРАММИРОВАНИЕ НА ЯЗЫКАХ С и С++: ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Учебное пособие

Воронеж 2009

ГОУВПО «Воронежский государственный
технический университет»

Ю.Э. Корчагин

ПРОГРАММИРОВАНИЕ НА ЯЗЫКАХ C и C++: ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Утверждено Редакционно-издательским советом
университета в качестве учебного пособия

Воронеж 2009

УДК 621.3

Корчагин Ю.Э. Программирование на языках С и С++: лабораторный практикум: учеб. пособие / Ю.Э. Корчагин. Воронеж: ГОУВПО «Воронежский государственный технический университет», 2009. 251 с.

В лабораторном практикуме рассматриваются теоретические и практические сведения, необходимые для овладения навыками программирования на языках С и С++. Практикум состоит из 7 лабораторных работ. В описании каждой лабораторной работы приведены теоретические сведения и индивидуальные задания.

Издание соответствует требованиям Государственного образовательного стандарта высшего профессионального образования по направлению 210300 “Радиотехника”, специальности 210302 “Радиотехника”, дисциплине “Информатика”.

Учебное пособие предназначено для студентов специальности 210302 “Радиотехника” очной формы обучения (в том числе по сокращенной программе), изучающих дисциплину “Информатика”.

Табл. 17. Ил. 30. Библиогр.: 19 назв.

Научный редактор профессор Г.В. Макаров

Рецензенты: кафедра сетей связи и систем коммутации Международного института компьютерных технологий (зав. кафедрой канд. техн. наук, доц. Р.Н. Андреев);
канд. физ.-мат. наук Е.В. Литвинов

ВВЕДЕНИЕ

Ни для кого не секрет, что значительная часть современных радиотехнических устройств построена на базе микропроцессорной техники. Самыми яркими примерами являются мобильные телефоны, пейджеры, MP3-плейеры, автомобильные сигнализации, устройства управления инжекторами и т.д. Разрабатываются многочисленные устройства, работающие совместно с прочно вошедшими в повседневную жизнь персональными компьютерами. Поэтому современному радиоинженеру жизненно необходимо владеть навыками программирования.

Среди разработчиков радиоаппаратуры общепринятым стандартом при написании программ является применение языков C и C++. Существует много книг о C/C++. Однако структура изложения материала в них, как правило, отличается от принятой при изучении программирования в ВУЗе. По-видимому, это связано с противоречием между необходимостью достаточно строгого изложения материала в лекции и его понятного разъяснения (зачастую с опережением) на практических занятиях. Заметное влияние на процесс обучения программированию оказывает специфика самого языка C++. С одной стороны, это язык, изучаемый постепенно. С другой стороны, овладение практическими навыками программирования требует опережающего изучения, а также возврата и переосмысления уже изученного.

Важнейшую роль при обучении играет лабораторный практикум. Считается, что научиться программировать можно только в результате практической деятельности. Поэтому написание этой книги продиктовано необходимостью структурировать изложение материала оптимальным для выполнения лабораторных работ способом. Книга состоит

из введения, двенадцати глав и заключения. Главы 1, 3, 4, 6, 10 являются теоретическими. Их цель — доведение до читателя минимальных сведений, необходимых для выполнения лабораторных работ. Вместе с их изучением необходимо пользоваться книгами с полным систематическим изложением языка C/C++. Главы 2, 5, 7, 8, 9, 11, 12 помимо теоретических сведений содержат описания семи лабораторных работ, примеры выполнения заданий и сами задания.

Хочется обратить особое внимание читателей на то, что в данном пособии намечена канва изучения, даны ссылки на более объёмные источники, сформулированы задания и намечены пути их решения. Использования только этой книги *недостаточно* для формирования устойчивого навыка программирования. Следует обязательно расширять теоретические знания, используя книги [1] – [19] из библиографического списка.

Желаю успехов !!!

1. ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ

1.1. Система команд процессора, машинный код, языки программирования

Любая микропроцессорная система состоит как минимум из двух компонент: процессора и памяти. Процессор представляет собой устройство, обеспечивающее работу по заданной программе и координирующее работу микропроцессорной системы в целом. Обобщённый алгоритм работы процессора описан Джоном фон Нейманом и изображён на рис. 1. Здесь обозначено: 1 — операция чтения команды из

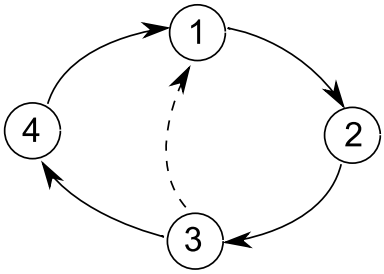

Рис. 1. Цикл фон Неймана

памяти, 2 — увеличение на единицу счетчика команд, 3 — расшифровка команды и принятие решения, необходимо ли ещё чтение из памяти, 4 — выполнение команды.

В зависимости от своей конструкции процессор имеет возможность выполнять некоторый конечный набор команд. Каждой команде ставят в соответствие её уникальный номер (код). Совокупность команд, выполняемых процессором, называют «системой команд процессора». Программа состоит из команд, которые содержат коды операций, информацию

об операндах (аргументах) и о том, куда поместить результат.

С точки зрения скорости работы программы, а также эффективного использования ресурсов процессора наиболее эффективным является программирование с использованием кодов команд процессора (программирование в машинном коде). Именно так составляли программы на заре развития вычислительной техники. Однако программирование в машинном коде — это очень трудоёмкий процесс.

Для облегчения программирования и некоторой стандартизации ввели в употребление языки программирования. Они позволяют писать программу в некотором, близком к человеческому языку виде. Так, например, каждой команде процессора можно поставить в соответствие буквенное или буквенно-цифровое обозначение и писать программу с использованием этих обозначений. Такой способ мало отличается от программирования в машинных кодах и получил название языка низкого уровня Ассемблера. Термин «низкого уровня» означает близость конструкций языка к машинному коду. Поскольку язык Ассемблера тесно связан с системой команд процессора, для каждого типа микропроцессорной системы существует свой диалект языка.

Заметное облегчение труда программиста принесло появление языков высокого уровня, где программа записывается словами английского языка, а её логика приближена к мышлению человека. Языки высокого уровня меньше учитывают особенности архитектуры компьютера. Поэтому программы, написанные на таких языках, легче переносятся с одной платформы на другую.

1.2. Трансляторы

Программа, написанная на любом из языков программирования, низкого или высокого уровня, непригодна для выполнения на ЭВМ. Предварительно её нужно перевести в машинный код. Если для перевода программы в машинный код используются некие технические средства, то применение языков программирования приводит к выигрышу в трудоёмкости написания программы.

Поэтому для каждого языка разрабатывается программа перевода текста программы с языка высокого уровня в машинный код. Эта программа называется «транслятор» и относится к классу инструментального программного обеспечения.

Трансляторы подразделяются на два вида: компиляторы и интерпретаторы. Компилятор преобразует исходный текст программы в исполняемую форму и сохраняет его в памяти или на внешнем запоминающем устройстве в виде исполняемого модуля (например, файл с расширением exe). После трансляции исполняемый файл можно запустить на выполнение самостоятельно, без участия компилятора. Процесс компиляции на самом деле состоит из двух частей — сама компиляция и компоновка. Результатом компиляции является так называемый объектный код (это машинный код, не привязанный к конкретным адресам памяти ЭВМ). Компоновка из объектного кода создаёт исполняемый машинный код.

Трансляторы другого типа (интерпретаторы) переводят исходный текст программы в команды микропроцессора частями и сразу организуют их выполнение. Программа под управлением интерпретатора выполняется медленней, но вместе с тем его использование позволяет эффективно

отлаживать программу, находить в ней ошибки.

При трансляции в первую очередь проверяется грамотность написания программы (синтаксис), и в случае отклонения от правил трансляция прерывается, выводится сообщение об ошибке. Синтаксически правильно написанная программа, но не отвечающая задуманному алгоритму, содержит логические ошибки, которые не в состоянии выявить транслятор. Исправление логических ошибок — самый трудоёмкий этап написания программы, для облегчения которого используют специальные программы, называемые отладчиками. Отладчики представляют собой набор средств, позволяющих контролировать ход выполнения разрабатываемой программы, просматривать содержимое ячеек памяти, регистров микропроцессора, выполнять программу по шагам, обозначать точки останова и т.д.

1.3. Алфавит языка

Все языки программирования являются так называемыми «искусственными языками», то есть разработанными людьми с какой-либо целью. Подобно «естественным языкам» они имеют свой алфавит, синтаксис и семантику.

К алфавиту языков C и C++ относятся [3]: буквы латинского алфавита и символ подчёркивания, цифры, специальные символы `. , ; : / { } \ < > ~ - = ? + ^ ! * % ' ( $ ) # | [ " ] @ &` и так называемые управляющие последовательности, используемые при взаимодействии с устройствами ввода и вывода. Заглавные и строчные буквы в языках C и C++ различаются. Буквы русского алфавита можно использовать в текстах и комментариях. Операторы языка отделяются друг от друга точкой с запятой.

1.4. Этапы создания программы

В процессе написания программы можно выделить следующие этапы:

1) Подготовка текста программы в одном их текстовых редакторов.

2) Трансляция программы.

3) Тестирование программы и её отладка.

Рассмотрим подробнее перечисленные этапы применительно к языкам C и C++.

1.4.1. Подготовка текста программы

Для этого этапа можно использовать любой текстовый редактор, который не сохраняет информацию о форматировании текста. Например, в среде Windows подойдёт текстовый редактор «Блокнот», а в Linux — редактор, встроенный в программу Midnight Commander. Следует заметить, что файл с программой на языке C должен иметь расширение «c», а на языке C++ — расширение «cpp». Чтобы контролировать правильность расширения имени файла в среде Windows необходимо снять галку «Скрывать расширения для зарегистрированных типов файлов» (окно «Проводника», пункт меню «Сервис», «Свойства папки», закладка «Вид»).

При использовании операционной системы Linux подготовить текст программы можно с помощью следующей последовательности действий.

1) Запустить командную строку (терминал).

2) Запустить файловый менеджер Midnight Commander командой mc.

3) Раскрыть папку, в которой будет храниться программа, например, /home/user/prog.

4) Создать новый текстовый файл с помощью комбинации клавиш Shift+F4. При этом запускается встроенный в Midnight Commander текстовый редактор.

5) Написать текст программы.

6) Выходя из текстового редактора (F10 или Esc) сохранить написанную программу.

Если требуется редактирование программы, то запустить встроенный в Midnight Commander текстовый редактор можно с помощью клавиши F4.

1.4.2. Трансляция программы

Процесс трансляции программы зависит от используемого транслятора. Многие из них запускаются из командной строки и в качестве входных данных используют файлы с текстом программы. В результате формируются файлы объектного и исполняемого кодов. Одними из лучших считаются свободно распространяемые трансляторы GCC для языка C и G++ для языка C++. Эти трансляторы доступны почти для всех операционных систем. Для компиляции необходимо использовать команды вида

```
gcc Имя_файла_программы,
g++ Имя_файла_программы,
```

в результате чего происходит компиляция, а затем компоновка. Здесь и далее команды, вводимые с клавиатуры и листинги программ будут выделяться шрифтом и зачастую напечатаны в отдельной строке. Однако нужно иметь в виду, что команды и программы в тексте книги являются частями предложений и согласно правилам русского языка требуют адекватной пунктуации. Поэтому приведённые выше команды вводятся без последних запятых и завершаются нажатием Enter. Если программа не содержит ошибок и трансля-

ция выполнена успешно, то в текущем каталоге будет создан исполняемый файл. По умолчанию в операционной системе Windows создается исполняемый файл с именем a.exe, а в системе Linux — a.out. Компиляторы gcc и g++ позволяют использовать ключ

```
-o Имя_выполняемой_программы,
```

например,

```
g++ -o programma.exe  main.cpp.
```

При написании больших серьёзных программ нецелесообразно располагать исходный код программы в одном файле. Чаще всего исходный код разбивают на логически законченные блоки, расположенные в разных файлах. Это позволяет компилировать отдельно разные части программы и получать объектные коды частей. Затем объектные коды частей программы совместно с библиотечными объектными кодами компонуются в исполняемый файл программы. Поэтому компилятор gcc имеет опцию -c, которая заставляет его отказаться от компоновки и создать только объектный код. Например, команда

```
g++ -c  main.cpp
```

приведёт к появлению в текущем каталоге файла main.o с объектным кодом. Для получения исполняемого файла можно использовать команду gcc (g++), которой в качестве входных данных необходимо указать список файлов объектных кодов, например,

```
g++ main.o.
```

В результате компилятор сам поймёт, что требуется компоновка и выполнит её.

Если в тексте программы присутствуют синтаксические ошибки, то компилятор выдаёт сообщение с указанием типа ошибки и номера строки, в которой она обнаружена, например,

```
main.cpp:7: error: ...
```

и прерывает процесс трансляции. Исполняемый файл программы не будет создан, пока не будут устранены все синтаксические ошибки.

1.4.3. Тестирование программы и её отладка

Отсутствие синтаксических ошибок в тексте программы позволяет успешно откомпилировать её. Однако алгоритм работы программы может содержать логические ошибки. Поэтому после компиляции необходимо тестировать работу программы. Запустить исполняемый файл на выполнение можно традиционным способом. Следует только отметить особенность запуска программы в ОС Linux: при запуске из командной стоки следует вначале использовать точку и прямой слеш:

```
./Имя_исполняемого_файла Enter.
```

Если в процессе тестирования программа ведёт себя непредсказуемо, например, выдаёт неправильный ответ для некоторых или всех тестовых входных данных, то необходимо искать логические ошибки в алгоритме работы программы. При этом можно пользоваться программами-отладчиками, рассмотрение которых выходит за рамки данной книги.

1.5. Интегрированные среды программирования

Процесс создания программы, описанный в предыдущем пункте не требует никаких дополнительных средств программирования кроме транслятора, однако, является достаточно трудоёмким. Вместе с тем существуют инструментальные средства, позволяющие облегчить и автоматизировать процесс создания программы. Обычно их называют интегрированными средами программирования. Многие из них включают в себя транслятор и дополнительные библиотеки исходных текстов и объектных кодов. Рассмотрим некторые из них.

1.5.1. Интегрированная среда Code::Blocks

Пакет программ Code::Blocks является свободно распространяемой средой разработки приложений. Code::Blocks существует как для ОС Windows, так и для Linux. Главное окно программы показано на рис. 2. Цифрой 1 на рис. 2 обозначена область менеджера проектов, цифрой 2 — область редактора исходного текста программы и цифрой 3 — область вывода служебных сообщений.

Создание проекта.

Для создания нового проекта в Code::Blocks следует воспользоваться пунктом меню «File, New, Project». Появится диалоговое окно, изображённое на рис. 3. Здесь необходимо выбрать шаблон, на основе которого будет создаваться проект. Программы, использующие для общения с пользователем интерфейс командной строки называют консольными приложениями. Им соответствует шаблон «Console application». После нажатия на кнопку «Go» предлагается выбрать язык (C или C++), а затем имя проекта и папку

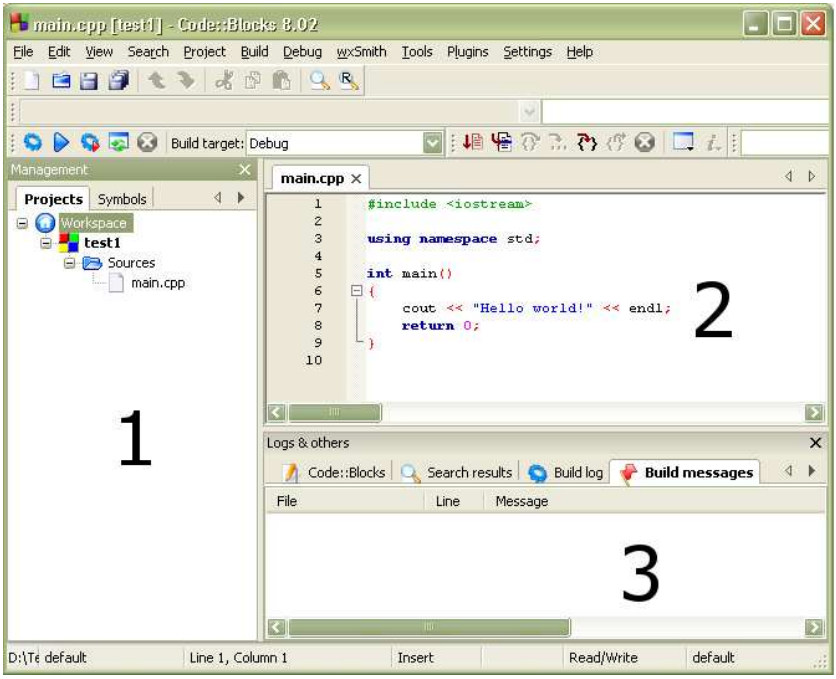

Рис. 2. Главное окно Code::Blocks

его расположения.

Последнее диалоговое окно изображено на рис. 4, рассмотрим его содержимое подробнее. Раскрывающийся список Compiler необходим для выбора одного из возможных трансляторов проекта (по умолчанию GNU GCC Compiler). Галочки «Create "Debug" configuration» и «Create "Release" configuration» позволяют создать цели для трансляции (сборки) проекта. Проект может компилироваться с целью тестирования и отладки, с целью изменения или обновления его компонентов, для окончательной готовности и т.д. В первом случае в исполняемый код программы вставляются подпрограммы взаимодействия с отладчиком. Для успешной ком-

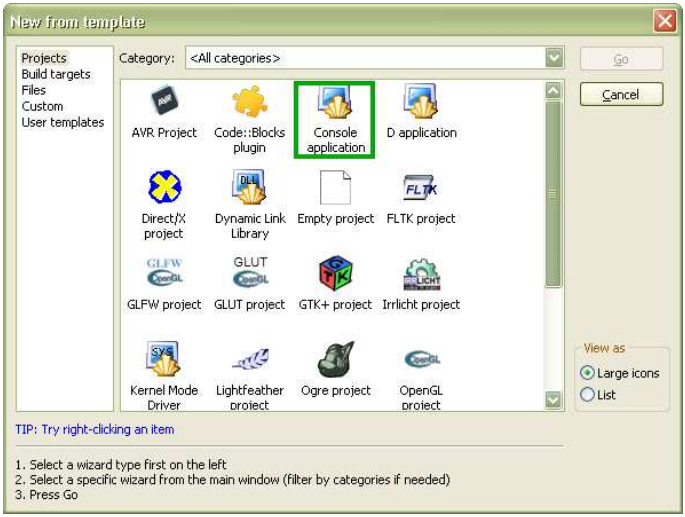

Рис. 3. Окно выбора шаблона проекта

пиляции должна существовать хотя бы одна цель. Управлять имеющимися целями можно в окне свойств проекта (меню «Project, Properties...», закладка «Build targets»). Если в окне, изображенном на рис. 4, оставить включенными обе галочки Debug и Release, то в свойствах проекта будут существовать две цели для компилятора.

После нажатия кнопки «Finish» интегрированная среда Code::Blocks создаст в папке проекта несколько файлов. Среди них находится файл описания проекта ИмяПроекта.cbp и файл с текстом программы main.cpp. Последний файл main.cpp содержит заготовку текста программы

```
#include <iostream>

using namespace std;
```

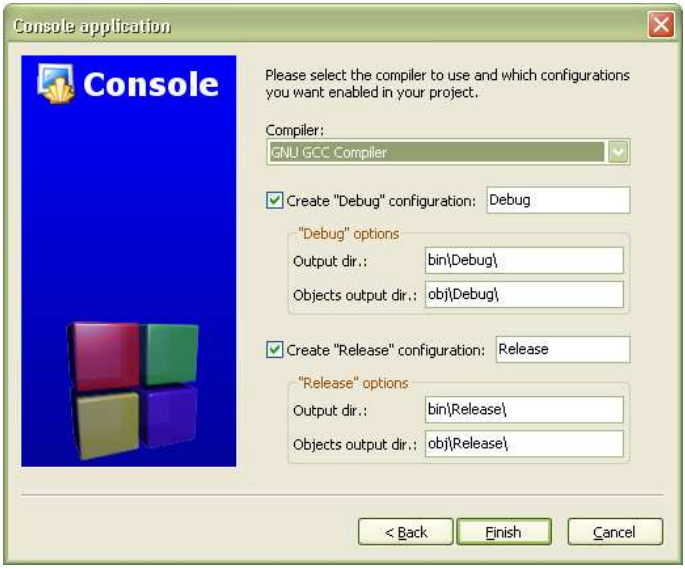

Рис. 4. Окно выбора компилятора

```
int main()
{
    cout << "Hello world!" << endl;
    return 0;
},
```

смысл которой станет понятен позже. Именно в этом файле следует писать текст программы!

Компиляция.

Перед компиляцией полезно сохранить проект и проконтролировать, какая выбрана цель сборки (меню «Project, Properties...», закладка «Build targets»).

Для компиляции необходимо использовать пункт меню «Build, Build» или комбинацию клавиш Ctrl+F9. Если

программа не содержит ошибок и компиляция выполнена успешно, то в каталоге проекта будут созданы две папки с именами obj и bin. Внутри каждой из них находятся папки с именами целей сборки. В папке obj после трансляции находится объектный код для каждого файла проекта, в папке bin — исполняемые файлы.

Запустить готовую программу на выполнение можно либо независимо от Code::Blocks, либо под управлением встроенного в Code::Blocks отладчика (пункты меню «Run Ctrl-F10» и «Build and run F9»).

1.5.2. Интегрированная среда Kdevelop

В ОС Liunx наиболее популярной заслуженно является интегрированная среда программирования Kdevelop. С точки зрения принципов работы Kdevelop довольно похож на среду Code::Blocks. Главное окно Kdevelop показано на рис. 5.

Создание проекта.

Для создания нового проекта в Kdevelop следует воспользоваться пунктом меню «Проект, Создать». Запускается мастер создания проектов, первое диалоговое окно которого изображено на рис. 6. Здесь необходимо выбрать шаблон, на основе которого будет создаваться проект, а также имя проекта и папку его расположения. Для создания обычного консольного приложения необходимо выбрать шаблон «Simple Hello World program». В следующих диалоговых окнах мастера создания проектов предлагается указать автора проекта, тип лицензии будущего приложения, нумерацию версий и т.д., что несущественно при обучении программированию. В результате работы мастера в каталоге расположения будет создана структура файлов и папок проекта. Файл с ис-

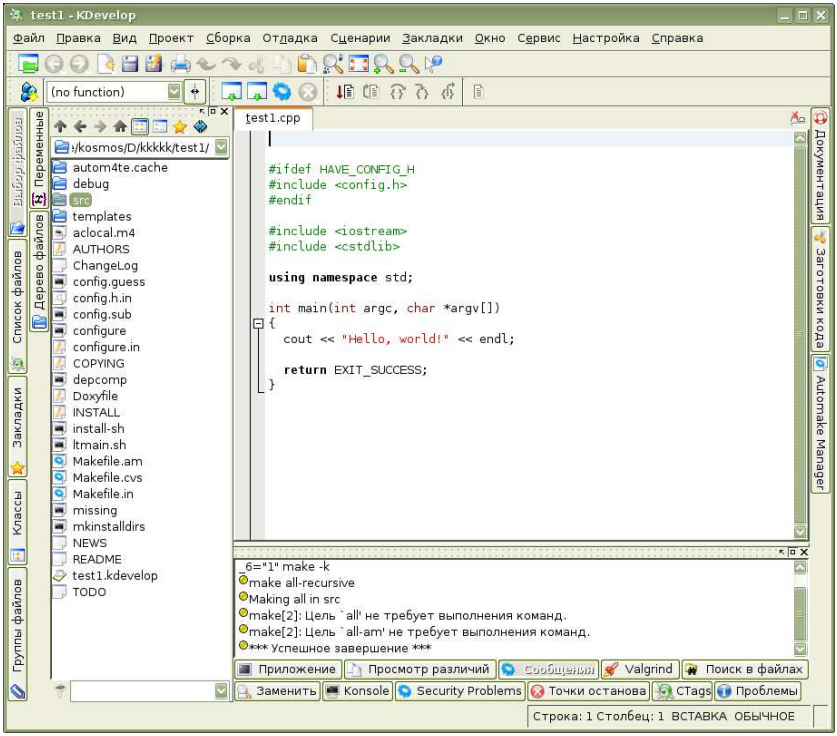

Рис. 5. Главное окно Kdevelop

ходным текстом «ИмяПроекта.cpp» программы находится в папке src и содержит заготовку текста программы

```
#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <iostream>
#include <cstdlib>

using namespace std;
```

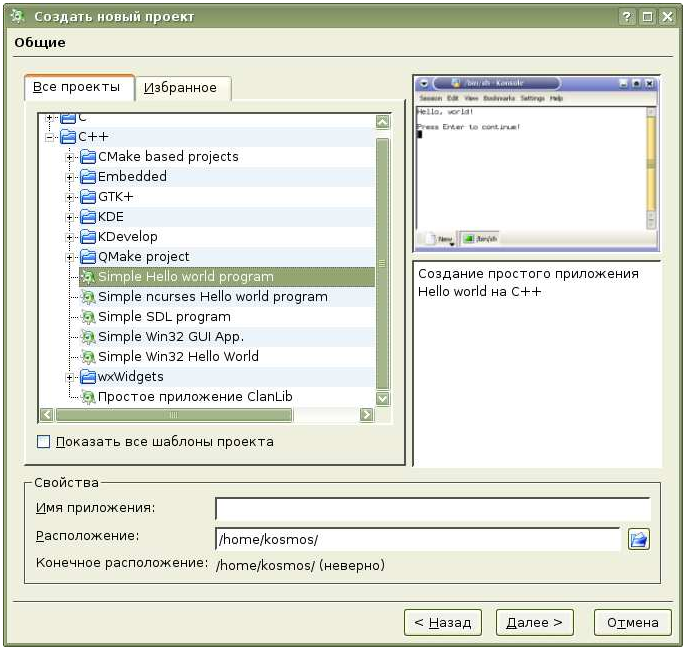

Рис. 6. Окно выбора шаблона проекта

```
int main(int argc, char *argv[])
{
  cout << "Hello, world!" << endl;
  return EXIT_SUCCESS;
}.
```

Компиляция.

Для компиляции проекта в Kdevelop необходимо использовать пункт меню «Сборка, Собрать» или функциональную клавишу F8. Kdevelop при компиляции тоже учитывает наличие целей сборки проекта, которые хранятся в

файле с именем Makefile в каталоге проекта. Поэтому при первой компиляции Kdevelop предлагает создать Makefile автоматически.

Если программа не содержит ошибок и компиляция выполнена успешно, то в каталоге проекта будут созданы две папки с именами obj и bin. Внутри каждой из них находятся папки с именами целей сборки. В папке obj после трансляции находится объектный код для каждого файла проекта, в папке bin — исполняемые файлы.

Запустить готовую программу на выполнение можно либо независимо от Kdevelop, либо под управлением встроенного в Kdevelop отладчика (пункт меню «Сборка, Выполнить приложение, Shift+F9»).

2. ЛАБОРАТОРНАЯ РАБОТА №1 ПРОГРАММИРОВАНИЕ ОСНОВНЫХ АЛГОРИТМИЧЕСКИХ КОНСТРУКЦИЙ НА ЯЗЫКАХ С и С++

Цели работы:

- овладение практическими навыками работы с компилятором С/С++;
- овладение практическими навыками программирования линейных, разветвляющихся и циклических алгоритмических конструкций.

2.1. Краткое описание языков С и С++

Язык программирования С был разработан в начале семидесятых годов XX века. К настоящему времени он претерпел несколько этапов своего развития. Язык программирования С++ объединил в себе достоинства языка С и возможности объектно-ориентированного программирования. К основным достоинствам языков С и С++ следует отнести гибкость и надёжность, переносимость программ с одной платформы на другую и с одной операционной системы на другую. Языки С и С++ являются языками высокого уровня. Но вместе с тем они обладают богатыми возможностями низкоуровневого программирования. Это делает их очень гибкими и пригодными как для начинающих пользователей, так и для профессиональных программистов.

2.1.1. Структура программы

Структура программы на языке С (С++) довольно произвольная. Обычно вначале программы располагаются так называемые директивы препроцессору. Каждая директива

начинается знаком #. Одна из директив называется include и служит для включения в текст программы ссылок на библиотеки языка.

Так, например, строка

```
#include <stdio.h>
```

включает в текст программы файл stdio.h, в котором расположены заголовки библиотечных функций из библиотеки stdio. Расширение файла «h» свидетельствует о том, что файл заголовочный. Поскольку stdio является стандартной библиотекой языка, имя файла stdio.h заключено в угловые скобки. В противном случае имя файла необходимо заключать в двойные кавычки.

С-программа состоит из подпрограмм, которые могут вызывать друг друга. Каждая подпрограмма называется функцией. Связь между функциями организована с помощью аргументов функций и возвращаемых ими значений. Среди всех функций одна является главной. Именно с этой функции, имя которой `main`, начинается выполнение программы. Функция `main` может иметь аргументы, с помощью которых происходит её взаимодействие с операционной системой. Параметры, указанные в командной строке при запуске программы, являются аргументами функции `main`. С помощью возвращаемого значения функция `main` может сообщить операционной системе о результате своей работы. Как правило возвращаемое значение, равное нулю, свидетельствует об успешном завершении программы. Ненулевое значение говорит о наличии ошибок и может быть проанализировано операционной системой.

Рассмотрим теперь классический пример простейшей программы, которая выводит поздравительное сообщение на экран.

```
#include <stdio.h>
int main(void)
{
printf(‘‘Поздравляю!\n’’);
return 0;
}
```

Строка `int main(void)` описывает функцию `main`. Служебное слово `void` свидетельствует о том, что функция `main` не требует аргументов. Слово `int` говорит о том, что функция `main` возвращает операционной системе целое значение. Само это значение указано после оператора `return` в конце программы. Тело функции заключено в фигурные скобки. Таким образом, общая схема описания любой функции выглядит так:

```
ТипРезультата ИмяФункции(СписокАргументов)
{
Тело функции;
}.
```

Точка в конце обозначает конец предложения и не относится к программе.

В рассмотренной программе использована также стандартная функция вывода на экран `printf`. В данном примере она выводит на экран слово «Поздравляю!» Символы `\n` необходимы для того, чтобы после вывода поздравительного сообщения курсор командной строки перешёл на следующую сроку. Оператор `return` осуществляет выход из функции и возврат значения, в нашем примере нулевого.

Идентификаторы.

Идентификатором является последовательность букв, цифр и знаков подчёркивания, которая начинается с бук-

вы или символа подчёркивания и не содержит пробелов [3]. Идентификаторы выступают в качестве имён каких-либо объектов. Символы нижнего и верхнего регистров в языке C (C++) считаются различными. Идентификатор может иметь произвольную длину, однако, значащими являются не все символы. Их число различно для разных систем программирования, а также может настраиваться. Идентификаторы не должны совпадать с ключевыми словами языка и именами стандартных библиотечных функций.

Комментарии.

Комментарии к программе можно записывать в любом месте программы. Они обрамляются символами /* Это комментарий */. В языке C++ доступна ещё одна форма комментария. Всё что написано после знаков // и до конца строки считается комментарием.

Переменные.

Программа, как правило, оперирует с некоторыми данными. Для их хранения в языках программирования используют переменные. Под переменной понимают именованную ячейку или область памяти. Переменные отличаются типом хранимых данных, именем, значением. Различные типы данных требуют различного размера выделяемой памяти, а также различной структуры этой памяти. Поэтому перед использованием переменной её нужно описать (объявить). При объявлении необходимо указать как минимум её тип и имя, например:

```
int i,j,k;
float x,y; .
```

Здесь объявлены три переменные целого типа с именами `i,j,k` и две переменные вещественного типа `x` и `y`. При объявлении можно указать значение переменной, например:

```
int i=1,j=10,k=500;
float x=3.14,y; .
```

Таким образом, объявления переменных в общем виде можно записать как

```
ТипПеременной СписокПеременных;
   или
ТипПеременной Имя1=Знач1,Имя2=Знач2,...; .
```

Важное значение имеет вопрос о месте объявления переменных. Одни могут быть объявлены вне каких-либо функций. Такие переменные называются глобальными. Они доступны для использования в любом месте программы, в любой функции. Другие переменные могут объявляться внутри какого-либо блока, например, внутри функции. Такие переменные называют локальными. Их можно использовать только внутри этого блока. Никакая функция не может использовать переменные, объявленные внутри другой функции. Поэтому в разных блоках можно использовать переменные с одинаковыми именами, хотя по смыслу это будут разные переменные.

2.1.2. Простые базовые типы

Типы данных всех языков программирования можно разделить на простые и структурированные. Переменная простого типа хранит одно значение. С языке C (C++) это целые и вещественные числа, символы и указатели. Переменные структурированных типов хранят много значений

простых или тоже структурированных типов. К таким типам относятся массивы, структуры, объекты и т.д. В языке С введены 5 простых типов, которые называют базовыми.

1. Переменная типа **char** занимает в памяти 1 байт. Её значениями могут быть символы кодовой таблицы.
2. Переменная типа **int** используется для хранения целых чисел и имеет длину 2 байта.
3. Для использования вещественных данных предусмотрен тип **float** длиной 4 байта.
4. Тип **double** соответствует вещественным числам двойной точности. Переменная такого типа занимает 8 байт памяти.
5. Слово **void** используется для объявления объекта, не имеющего значения, например, при описании функции, не возвращающей никакого значения или не имеющей аргументов.

Перечисленные базовые типы могут быть использованы совместно с модификаторами, которые ставятся перед названием типа: **signed** — знаковый, **unsigned** — беззнаковый, **long** — длинный, **short** — короткий.

Целые типы.

Переменные одного из целых типов предназначены для хранения целых чисел со знаком или без знака (только положительные и нуль). Базовым целым типом является тип `int`, для которого модификаторы `signed` и `short` являются модификаторами по умолчанию. Таким образом, для объявления переменной M целого типа равносильны следующие описания

```
int M;
short int M;
```

```
signed int M;
signed short int M; .
```

Переменная типа `int` занимает 2 байта памяти и может принимать значения от -32768 до 32767. Беззнаковые целые переменные `unsigned int` также занимают 2 байта памяти, но могут принимать значения от 0 до 65535.

Очевидно, что в программе может потребоваться хранить данные, которые не укладываются в диапазон значений, предоставляемый целым типом. В этом случае целесообразно использовать длинный целый тип `long int`. Переменные этого типа занимают по 4 байта памяти и могут принимать значения от -2147483648 до 2147483647. Беззнаковый длинный целый тип `unsigned long int` предусматривает для хранения данных 4 байта и диапазон значений от 0 до 4294967295. Характеристики перечисленных целых типов приведены в табл. 1.

Таблица 1

Целые типы

Название	**Идентификатор**	**Диапазон значений**	**Размер памяти**
Знаковый целый	int signed int short int signed short int	-32768.. ..32768	2 байта
Беззнаковый целый	unsigned int unsigned short int	0..65535	2 байта
Длинный знаковый целый	long int signed long int	-2147483648 .. 2147483647	4 байта
Беззнаковый длинный целый	unsigned long int	0..4294967295	4 байта

Вещественные типы.

В группу вещественных типов входят три типа: `float`, `double` и `long double`. Они отличаются количеством выделяемой памяти, а также диапазоном возможных значений. Характеристики вещественных типов приведены в табл. 2.

Таблица 2

Вещественные типы

Название	**Идентификатор**	**Диапазон значений по модулю**	**Размер памяти**
Вещественное одинарной точности	float	от $3.4 \cdot 10^{-38}$ до $3.4 \cdot 10^{38}$	4 байта
Вещественное двойной точности	double	от $1.7 \cdot 10^{-308}$ до $1.7 \cdot 10^{308}$	8 байт
Вещественное повышенной точности	long double	от $3.4 \cdot 10^{-4932}$ до $1.1 \cdot 10^{4932}$	10 байт

Символьный тип char.

Переменные и константы этого типа могут принимать значения из множества символов кодовой таблицы и занимают в памяти 1 байт. Символ, заключенный в апострофы, обозначает константу символьного типа, например: `'5'`, `'d'`. Однако в памяти хранится именно код символа из кодовой таблицы. Поэтому символы также можно интерпретировать как целые числа и применять к ним модификаторы `signed` и `unsigned`. По умолчанию для типа `char` используется модификатор `singed`. Следовательно значения переменных символьного типа интерпретируются как целые числа со знаком от -128 до 127, что иллюстрирует табл. 3.

Таблица 3

Символьные типы

Идентификатор	**Диапазон значений**	**Размер памяти**
char или signed char	-128..127	1 байт
unsigned char	0..255	1 байт

Символы с кодами от 0 до 31 являются управляющими символами (см. Приложение), они не соответствуют ни одному печатному символу, их нельзя ввести с клавиатуры. Для возможности использования в языке C (C++) предусмотрены управляющие константы, некоторые из которых перечислены в табл. 4.

Таблица 4

Управляющие константы

Последовательность	**Значение**
`\b`	BS, забой
`\f`	Новая страница, перевод страницы
`\n`	Новая строка, перевод строки
`\r`	Возврат каретки
`\t`	Горизонтальная табуляция
`\v`	Вертикальная табуляция
`\''`	Двойная кавычка
`\'`	Апостроф
`\\`	Обратный слеш
`\0`	Нулевой символ, нулевой байт
`\a`	Звуковой сигнал

2.1.3. Оператор присваивания

Для языка C++ проявления операции присваивания весьма многообразны. Простая форма операции присваивания задается знаком =, например, `pi=3.14;`. Помимо основной в языке C++ существуют специальные формы оператора присваивания, примеры которых показаны в табл. 5.

Таблица 5

Формы оператора присваивания

Обычная запись	**Более коротко на C++**
A = A * B	A *= B
A = A / (B + 4)	A /= B + 4
A= A % B	A % = B
A=A+B	A += B
A=A-B	A -= B

2.1.4. Арифметические и логические операции

В программировании для определения действий служат выражения (аналог математических формул). Выражения состоят из операций и операндов. Операции делятся на унарные (один операнд) и бинарные (два операнда). В табл. 6 перечислены унарные и бинарные операции, используемые в C++ чаще остальных.

Таблица 6

Операции C++

Унарные операции	
&	Операция взятия адреса
*	Операция обращения по ссылке
+	Унарный плюс
–	Унарный минус
~	Поразрядное дополнение (дополнение до единицы)
!	Логическое отрицание
++	Префикс: пред-инкремент, постфикс: пост-инкремент
– –	Префикс: пред-декремент, постфикс: пост-декремент
Бинарные операции	
+	Бинарный плюс (сложение)
–	Бинарный минус (вычитание)
*	Умножение
/	Деление
%	Остаток от деления
<<	Сдвиг влево, а также помещение в поток

Продолжение табл. 6

>>	Сдвиг вправо, а также извлечение из потока
&	Поразрядное И
|	Поразрядное включающее ИЛИ
&&	Логическое И
||	Логическое ИЛИ
=	Присваивание
*=	Присвоить произведение
/=	Присвоить частное
%=	Присвоить остаток
+=	Присвоить сумму
-=	Присвоить разность
<<=	Присвоить сдвиг влево
>>=	Присвоить сдвиг вправо
&=	Присвоить поразрядное И
^=	Присвоить поразрядное исключающее ИЛИ
|=	Присвоить поразрядное ИЛИ
<	Меньше
>	Больше
<=	Меньше или равно
>=	Больше или равно
= =	Равно
!=	Не равно

2.1.5. Вывод на экран

Общение с внешним миром программа на языке С реализует с помощью библиотеки функций ввода-вывода stdio. Для её использования необходимо включить в программу заголовочный файл библиотеки

```
#include <stdio.h> .
```

Одним из способов вывода на экран является использование функции printf:

```
printf ("Управляющая строка",Арг1,Арг2,...);
```

Управляющая строка может содержать компоненты трёх типов:
1) обычные символы, которые просто выводятся на экран;
2) управляющие константы из табл. 5;
3) спецификации преобразования.
Каждая спецификация преобразования служит для вывода на экран очередного данного из списка аргументов Арг1, Арг2, ... Начинается спецификация знаком %, заканчивается — символом преобразования. В следующем примере

```
printf("Значение числа Пи равно %f\n", pi);
```

символом преобразования является буква f. Следовательно функция `printf` выведет на экран вместо `%f` значение переменной `pi`, которая находится в списке аргументов.

Каждому аргументу должна соответствовать своя спецификация преобразования:
`%d` – десятичное целое число;
`%f` – вещественное число;
`%c` – символ;
`%s` – строка.

В языке C++ на основе объектно-ориентированного способа программирования реализован так называемый механизм потоковых объектов. В результате подключения к программе библиотеки функций для работы с потоковыми объектами

```
#include <iostream.h>
```

становится доступен объект выходного потока **cout**, связанный с экраном. Поэтому для вывода на экран достаточно выводимый объект направить в поток. Вот несколько примеров:

```
cout << "Поздравляем!!!\n";
cout << "Значение Пи равно" << pi << ‘‘\n’’;  //.
```

Здесь операция помещения в поток << выполняется цепочкой слева направо.

2.1.6. Ввод с клавиатуры

Для ввода значений переменных с клавиатуры служит функция scanf:

```
scanf ("Управляющая строка",Арг1,Арг2,...); .
```

Функция `scanf` работает аналогично функции `printf`, но управляющая строка должна состоять только из спецификаций преобразования. В качестве аргументов указывают имена переменных, которым будут присвоены введенные данные, с символом & вначале, например,

```
scanf("%d%f ", &x,&y);   //.
```

В языке C++ ввод с клавиатуры можно осуществить с помощью чтения из потокового объекта **cin**, связанного с клавиатурой, например,

```
cout << "Введите значение x";
cin >> x;
cout << "\n";  //.
```

2.1.7. Виды алгоритмов

По структуре все программы можно разделить на 3 основные группы: *линейные, разветвляющиеся и циклические.*

Линейным называется вычислительный процесс (алгоритм), в котором действия выполняются в линейной последовательности, одно за другим, и каждое действие выполняется один раз. Обычно такие алгоритмы состоят из команд присваивания.

Выполнение разветвляющегося алгоритма происходит по одной из нескольких заранее запланированных ветвей в зависимости от выполнения или невыполнения некоторого условия. К примеру, алгоритм решения квадратного уравнения зависит от знака дискриминанта.

В циклическом алгоритме одни и те же действия выполняются многократно. Такой многократно повторяющийся участок алгоритма называется циклом.

Пример.

Рассмотрим процесс создания простейшей программы с линейным алгоритмом на примере следующей задачи.

Напишите программу вычисления дохода по вкладу. Процентная ставка (в процентах годовых) и время хранения (в днях) задаются во время работы программы.

Приведем текст программы полностью, а затем обсудим её содержание.

```
#include <iostream.h>
float sum,      //сумма вклада
      srok,     //срок вклада
      stavka,   //процентная ставка
      dohod;    //доход по вкладу
int main(void)
{
cout << "Вычисление дохода по вкладу\n";
cout << "Введите исходные данные:\n";
cout << "Величина вклада (руб) ->";
cin >> sum;
cout << "\n";
cout << "Срок вклада (дней) ->";
cin >> srok;
cout << "\n";
```

```
cout << "Процентная ставка ->";
cin >> stavka;
cout << "\n";
dohod=(sum*stavka/100)*srok/365;
sum=sum+dohod;
cout << "\n";
cout << "-------------------------\n";
cout << "Доход:" << dohod << "руб\n";
cout << "Конечная сумма:" << sum << "руб\n";
} // конец программы.
```

Вначале программы строкой `#include <iostream.h>` включается заголовочный файл библиотеки `iostream` для использования потоковых объектов вывода на экран и ввода с клавиатуры. Далее объявляются переменные `sum`, `srok`, `stavka`, `dohod` вещественного типа `float`. Главная и единственная функция `main`, с которой начинается выполнение программы описана как `int main(void)`. Это означает, что функция не зависит ни от каких аргументов и может передать операционной системе целочисленный результат своей работы. Строки

```
cout << "Вычисление дохода по вкладу\n";
cout << "Введите исходные данные:\n";
```

выводят на экран пригласительную информацию. Выводимая строка заканчивается сочетанием `\n`. Это приводит к тому, что после вывода на экран курсор переходит на следующую строку, где и будет располагаться следующий вывод на экран или ввод данных.

Исходные данные пользователь будет вводить с клавиатуры. Для удобства сначала выводится подсказка

```
cout << "Величина вклада (руб) ->";
```

затем команда

```
cin >> sum;
```

осуществляет ввод с клавиатуры значения начальной суммы, положенной в банк. Введённое значение присваивается переменной `sum`. Строка

```
cout << "\n";
```

выводит на экран непечатный символ `\n`, в результате курсор переводится на следующую строку. Аналогично выполняется ввод срока вклада в днях

```
cout << "Срок вклада (дней) ->";
cin >> srok;
cout << "\n";
```

а также величины процентной ставки

```
cout << "Процентная ставка ->";
cin >> stavka;
cout << "\n";   //.
```

Введённые значения присваиваются переменным `srok` и `stavka` соответственно.

Расчёт дохода по вкладу выполняется по формуле

$$\text{dohod} = \text{sum} \cdot \frac{\text{stavka}}{100} \cdot \frac{\text{srok}}{365},$$

поэтому следующие две строки программы представляют собой операторы присваивания, первый из которых вычисляет доход, а второй изменяет значение суммы вклада:

```
dohod=(sum*stavka/100)*srok/365;
sum=sum+dohod;   //.
```

Результаты вычислений выводятся на экран:

```
cout << "\n";
cout << "------------------------\n";
cout << "Доход:" << dohod << "руб\n";
cout << "Конечная сумма:" << sum << "руб\n";   //.
```

2.1.8. Разветвляющиеся алгоритмы.

Для записи разветвляющихся алгоритмов в языке C (C++) предусмотрены два оператора ветвления **if** и **switch**.

Оператор `if` осуществляет ветвление по двум направлениям и в общем виде выглядит так:

```
if (Условие) Оператор1; else Оператор2;   //.
```

При выполнении оператора `if` сначала выполняется проверка условия. Если оно истинно, то выполняется `Оператор 1`, в противном случае — `Оператор 2`. Блок-схема, соответствующая оператору if изображена на рис. 7. В качестве условия может использоваться не только логическое выражение. Любое выражение, принимающее ненулевое значение, считается истинным, нулевое значение — ложным.

Допускается сокращенный вариант оператора if:

```
if (Условие) Оператор1;   //.
```

Тогда при истинности условия выполняется `Оператор 1`, в противном случае выполняется следующий после `if` оператор. Блок-схема соответствующая сокращенной форме оператора `if` изображена на рис. 8.

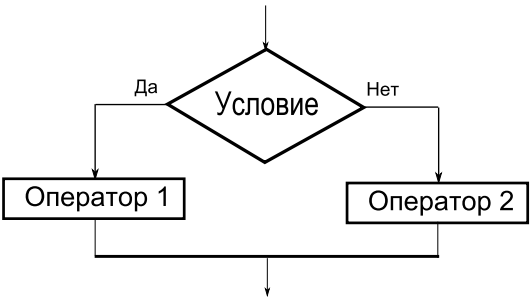

Рис. 7. Блок-схема разветвляющегося алгоритма

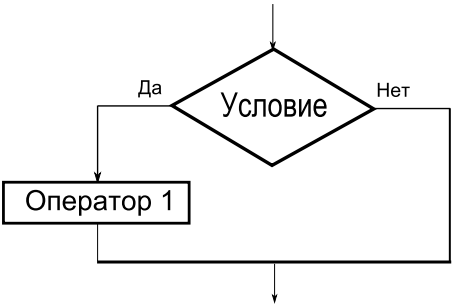

Рис. 8. Блок-схема сокращённого ветвления

Если по смыслу задачи вместо первого или второго оператора необходима последовательность из нескольких операторов, то их заключают в фигурные скобки $\{\ldots\}$:

```
{
Оператор11;
Оператор12;
...
Оператор1N:
};   //.
```

Полученная конструкция называется составным оператором.

Пример.

Даны длины трёх сторон треугольника. Если такой треугольник существует, вычислить его площадь.

Программа:

```
#include <iostream.h>
#include <math.h>
double a,b,c,p,s;
int main(void)
{
cout << "Введите длины сторон треугольника\n";
cin >> a >> b >> c;
if ((a+b<c)||(a+c<b)||(b+c<a))
cout << "Такого треугольника не существует\n";
else
{
p=(a+b+c)/2;
s=sqrt(p*(p-a)*(p-b)*(p-c));
cout << "Площадь треугольника равна" << s << "\n";
};
}    //.
```

Здесь использована логическая операция ИЛИ, обозначенная двумя вертикальными линиями, а также стандартная функция вычисления квадратного корня `sqrt` из библиотеки `math`.

2.1.9. Циклические алгоритмы

Любой циклический алгоритм состоит из следующих компонентов: подготовка цикла, проверка условия продолжения или окончания цикла, модификация условия и тело цикла (многократно выполняющиеся операторы). В зависимости от взаимного расположения компонент все циклы при-

нято делить на циклы с предусловием и постусловием. Если проверка условия находится перед телом цикла, то такой цикл относится к циклам с предусловием. Его блок-схема изображена на рис. 9. В противном случае цикл является циклом с постусловием. Его блок-схема показана на рис. 10.

Для организации циклов в языке C (C++) предусмотрены три оператора.

1) Оператор с предусловием **while**

```
while (Выражение) Оператор;  //.
```

Выражение в скобках — это условие продолжения цикла, Оператор — простой или составной — представляет собой тело цикла. В качестве условия продолжения может использоваться не только логическое выражение. Любое выражение, принимающее ненулевое значение считается истинным, нулевое значение — ложным. Однако в теле необходимо предусмотреть модификацию условия для того, чтобы цикл мог на каком-либо шаге закончиться. Проверка условия продолжения цикла осуществляется перед выполнением тела. Поэтому оператор while называется оператором цикла с предусловием. Его тело может ни разу не выполниться.

Пример.

Дано 100 вещественных чисел, вводимых с клавиатуры. Найти разность между максимальным и минимальным из них.

Поиск максимального и минимального значения из многих введённых данных осуществим следующим образом. Воспользуемся переменными `max` и `min` того же типа, что и вводимые данные. Первое введённое число является и максимальным и минимальным одновременно. Поэтому присвоим

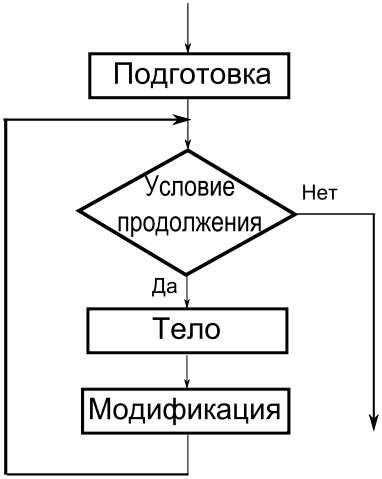

Рис. 9. Блок-схема цикла с предусловием

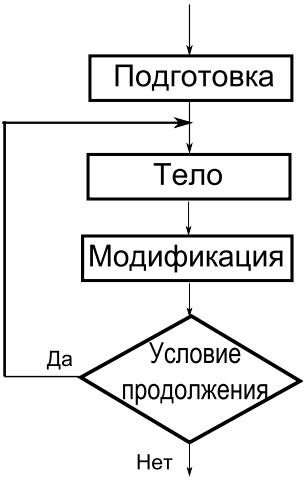

Рис. 10. Блок-схема цикла с постусловием

переменным max и min одинаковое значение, равное первому введённому числу. Далее многократно будем выполнять ввод очередного числа, сравнение его с max и min. Для этого используем оператор цикла. Чтобы пересчитать количество введённых чисел воспользуемся переменной j целого типа. Начальное значение переменной j выберем равным 1. В теле цикла значение переменной j будем увеличивать на 1, а условием продолжения цикла выберем неравенство $j < 100$. Ниже приведён текст программы, соответствующий описанному алгоритму, а на рис. 11 показана его блок-схема.

```
#include <iostream.h>
float x,min,max;
int j;
int main (void)
{
cout << ‘‘Вводите числа!’’;
cin >> x;
min=x; max=x;
j=1;
while (j<100)
{
cin >> x;
if (x<min) min=x;
if (x>max) max=x;
j++; // Это эквивалентно i=i+1
};
cout << ‘‘Искомая разность равна’’ << max-min << ‘‘\n’’;}
```

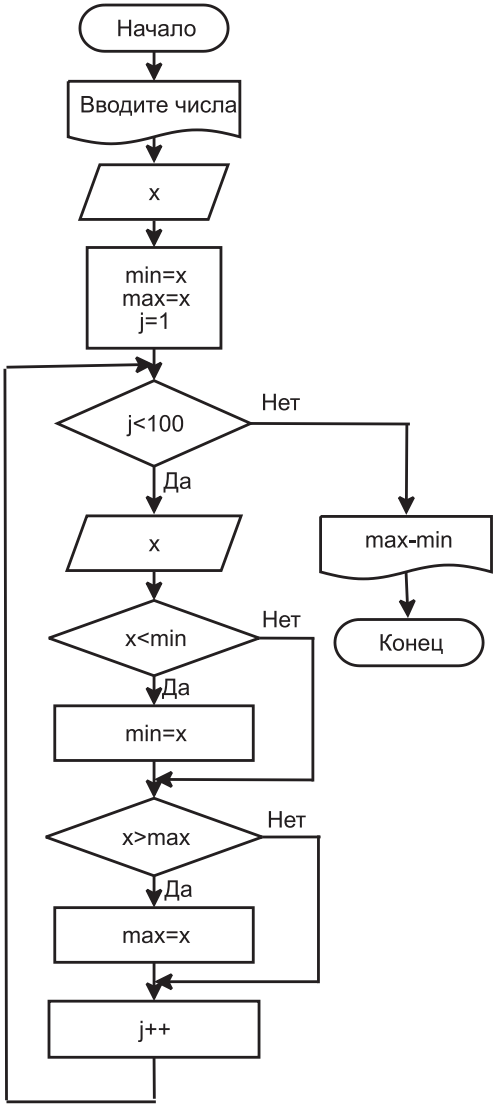

Рис. 11. Блок-схема решения задачи при использовании оператора цикла с предусловием

2) Оператор с постусловием **do while**

```
do
{Тело цикла}
while (Выражение);   //.
```

Выражение — это условие продолжения цикла. Тело цикла может состоять из большого числа операторов, среди которых необходимо предусмотреть модификацию условия для того, чтобы цикл мог на каком-либо шаге закончиться. Проверка условия продолжения цикла осуществляется после выполнения тела. Поэтому оператор цикла do ... while называется оператором цикла с постусловием. Тело такого цикла хотя бы один раз выполнится.

Пример.

Дано 100 вещественных чисел, вводимых с клавиатуры. Найти разность между максимальным и минимальным из них.

Приведём текст программы, основанный на предыдущем примере.

```
#include <iostream.h>
float x,min,max;
int j;
int main (void)
{
cout << "Вводите числа!";
cin >> x;
min=x; max=x;
j=1;
do
{cin >> x;
```

```
if (x<min) min=x;
if (x>max) max=x;
j++;}
while (j<100);
cout << "Искомая разность равна" << max-min << "\n";
}
```

Блок-схема решения задачи с использованием оператора цикла с постусловием изображена на рис. 12.

3) Оператор цикла с параметром применяется для организации циклов с известным количеством повторений и относится к циклам с предусловием. Формат оператора следующий:

```
For (Выражение1;Выражение2;Выражение3) Тело;  //.
```

В выражениях 1-3 должна фигурировать переменная цикла. Её значение изменяется в ходе выполнения цикла и по нему устанавливается необходимость повторения цикла или выхода из него. `Выражение 1` служит для присвоения начального значения переменной цикла. С помощью `Выражения 2` происходит проверка условия продолжения цикла, а `Выражение 3` измененяет значение переменной цикла, например,

```
for (i=1; i<=100;i++) {...}  //.
```

Пример.

Дано 100 вещественных чисел, вводимых с клавиатуры. Найти разность между максимальным и минимальным из них.

Вновь модифицируем текст программы предыдущего примера.

```
#include <iostream.h>
```

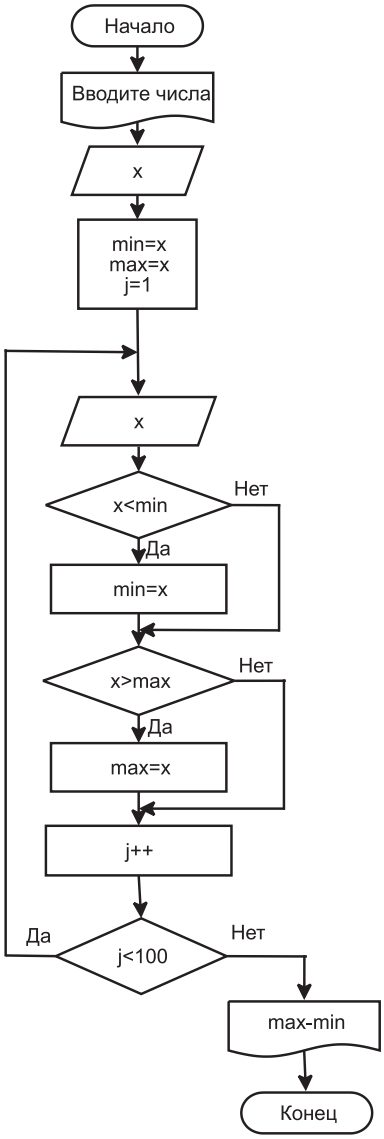

Рис. 12. Блок-схема решения задачи при использовании оператора цикла с постусловием

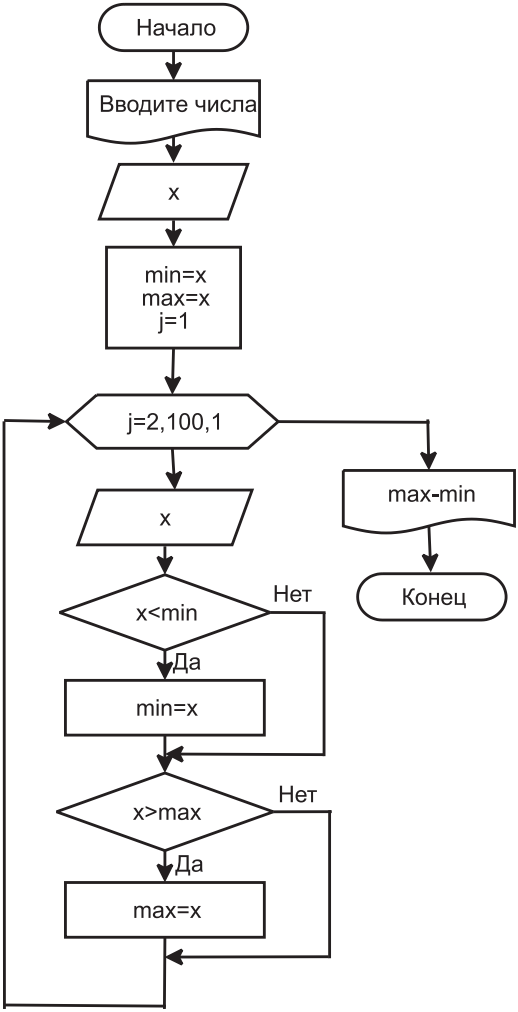

Рис. 13. Блок-схема решения задачи при использовании оператора цикла с параметром

```
foat x,min,max;
int j;
int main (void)
{
cout << ‘‘Вводите числа!’’;
cin >> x;
min=x; max=x;
j=1;
for (j=2; j<=100; j++)
{
cin >> x;
if (x<min) min=x;
if (x>max) max=x;
};
cout << ‘‘Искомая разность равна’’ << max-min << ‘‘\n’’;
}  //.
```

Поскольку для задания начальных значений переменным `min` и `max` первое число вводится вне цикла, переменная цикла `j` принимает значения от 2 до 100. Внутри тела цикла модификации переменной `j` не предусмотрено. Блок-схема решения задачи с использованием оператора цикла с параметром изображена на рис. 13.

2.1.10. Стандартные библиотечные функции

Перечислим отдельным пунктом наиболее часто употребляемые библиотечные функции (см. табл. 7). Более полную информацию о библиотечных функциях можно найти в приложениии книги [1] и в книге [13].

Таблица 7

Библиотечные функции

Имя функции	**Смысл**
Функции библиотеки math	
acos	Арккосинус
asin	Арксинус
atan	Арктангенс
cos	Косинус
sin	Синус
tan	Тангенс
cosh	Гиперболический косинус
sinh	Гиперболический синус
tanh	Гиперболический тангенс
exp	Экспонента
log	Натуральный логарифм
log10	Десятичный логарифм
pow	Возведение в степень pow(x,y)=x^y
sqrt	Квадратный корень
ceil	Округление до большего
fabs	Модуль вещественного числа
floor	Округление до меньшего
Функции библиотеки stdlib	
atof	Преобразование строки в число типа float
atoi	Преобразование строки в число типа int
atol	Преобразование строки в число типа long int
rand	Генерация случайного числа
abs	Модуль целого числа
labs	Модуль длинного целого

2.2. Практическое задание

При выполнении практического задания придерживайтесь следующей последовательности действий.

1. **Обязательно** изучите описание, предлагаемое выше.
2. Разберите примеры, приводимые в описании.
3. Осмыслите задачу, предлагаемую в Вашем варианте.

4. Составьте словесный алгоритм решения задачи.
5. Представьте алгоритм решения в виде блок-схемы.
6. Переведите алгоритм на язык программирования.

2.2.1. Варианты заданий

Вариант 1.

Для двух натуральных чисел P и Q <1000000 напишите программу, которая определяет, являются ли они взаимно простыми. Два числа называются взаимно простыми, если они не имеют общих делителей больших 1.

Вариант 2.

В обращении имеются монеты 1-,5-,10-,50-копеечного достоинства. Напишите программу, определяющую сдачу наименьшим количеством монет.

Вариант 3.

Натуральное число является совершенным, если оно равно сумме всех своих делителей, включая 1. Напишите программу, которая выводит на экран совершенные числа, меньшие 1000.

Вариант 4.

Натуральное число является простым, если оно не имеет делителей, кроме самого себя и единицы. Напишите программу, которая выводит на экран простые числа, меньшие 1000.

Вариант 5.

Исходными данными являются координаты точки M(x,y). Составьте программу, которая определяет, принад-

лежит ли точка M области пересечения окружностей, изображённой на рис. 14.

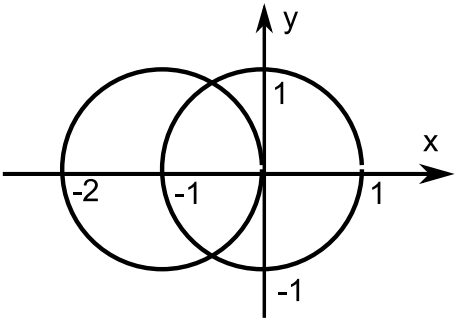

Рис. 14. Рисунок к варианту 5

Вариант 6.

Исходными данными являются координаты точки M(x,y). Составьте программу, которая определяет, принадлежит ли точка M фигуре, изображённой на рис. 15.

Вариант 7.

Составьте программу, вычисляющую значение выражения $S = \sqrt{3 + \sqrt{6 + \sqrt{9 + ... + \sqrt{96 + \sqrt{99}}}}}$.

Вариант 8.

Составьте программу, вычисляющую значение выражения

$$S = \cos\left(1 + \cos\left(2 + \cos\left(3 + ... + \cos\left(39 + \cos\left(40\right)\right) ...\right)\right)\right).$$

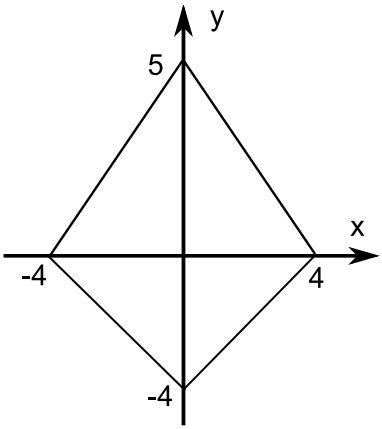

Рис. 15. Рисунок к варианту 6

Вариант 9.

Напишите программу, которая вычисляет количество точек с целочисленными координатами, находящихся в круге радиуса $R > 0$ с центром в точке A(x,y).

Вариант 10.

С клавиатуры вводится десятичное число M и основание системы счисления от 2 до 9. Составьте программу, которая переводит число M в указанную систему счисления.

Вариант 11.

Составьте программу расчёта суммы $1+2+3+4+...+N$. Полученный результат сравните с контрольным значением, вычисленным по формуле $N(N+1)/2$. Количество слагаемых N вводится с клавиатуры.

Вариант 12.

Составьте программу расчёта суммы $1 + 3 + 5 + 7 + ... + (2N - 1)$. Полученный результат сравните с контрольным значением, вычисленным по формуле N^2. Количество слагаемых N вводится с клавиатуры.

Вариант 13.

Составьте программу расчёта суммы $2+4+6+8+...+2N$. Полученный результат сравните с контрольным значением, вычисленным по формуле $N(N+1)$. Количество слагаемых N вводится с клавиатуры.

Вариант 14.

Составьте программу расчёта суммы $1^2+2^2+3^2+4^2+...+N^2$. Полученный результат сравните с контрольным значением, вычисленным по формуле $N(N+1)(2N+1)/6$. Количество слагаемых N вводится с клавиатуры.

Вариант 15.

Составьте программу расчёта суммы $1^2+3^2+5^2+7^2+...+(2N-1)^2$. Полученный результат сравните с контрольным значением, вычисленным по формуле $N(4N^2-1)/3$. Количество слагаемых N вводится с клавиатуры.

Вариант 16.

Составьте программу расчёта суммы $1^3+2^3+3^3+4^3+...+N^3$. Полученный результат сравните с контрольным значением, вычисленным по формуле $N^2(N+1)^2/4$. Количество слагаемых N вводится с клавиатуры.

Вариант 17.

Составьте программу расчёта суммы $1^3 + 3^3 + 5^3 + 7^3 + ... + (2N-1)^3$. Полученный результат сравните с контрольным значением, вычисленным по формуле $N^2(2N^2-1)$. Количество слагаемых N вводится с клавиатуры.

Вариант 18.

Составьте программу расчёта суммы $1^4 + 2^4 + 3^4 + 4^4 + ... + N^4$. Полученный результат сравните с контрольным значением, вычисленным по формуле $(N^2+N)(2N+1)(3N^2+3N-1)/30$. Количество слагаемых N вводится с клавиатуры.

Вариант 19.

Составьте программу приближённого расчёта бесконечной суммы

$$\sum_{i=1}^{\infty} i^{-2} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + ...$$

Суммирование необходимо проводить, пока модуль очередного слагаемого не станет меньше заранее заданного маленького числа $\epsilon = 10^{-4}$, характеризующего точность. Результат суммирования сравните с точным значением $\pi^2/6$. Вычислите погрешность и сопоставьте её с точностью.

Вариант 20.

Составьте программу приближённого расчёта бесконечной суммы

$$\sum_{i=1}^{\infty} (-1)^{i-1} i^{-2} = 1 - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + ...$$

Суммирование необходимо проводить, пока модуль очередного слагаемого не станет меньше заранее заданного маленького числа $\epsilon = 10^{-5}$, характеризующего точность. Результат суммирования сравните с точным значением $\pi^2/12$. Вычислите погрешность и сопоставьте её с точностью.

Вариант 21.

Составьте программу приближённого расчёта бесконечной суммы

$$\sum_{i=1}^{\infty}(2i-1)^{-2} = 1 + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + ...$$

Суммирование необходимо проводить, пока модуль очередного слагаемого не станет меньше заранее заданного маленького числа $\epsilon = 10^{-4}$, характеризующего точность. Результат суммирования сравните с точным значением $\pi^2/8$. Вычислите погрешность и сопоставьте её с точностью.

Вариант 22.

Составьте программу приближённого расчёта бесконечной суммы

$$\sum_{i=1}^{\infty}\frac{(-1)^{i+1}}{2i-1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + ...$$

Суммирование необходимо проводить, пока модуль очередного слагаемого не станет меньше заданного маленького числа $\epsilon = 10^{-4}$, характеризующего точность. Результат суммирования сравните с точным значением $\pi/4$. Вычислите погрешность и сопоставьте её с точностью.

Вариант 23.

Составьте программу приближённого вычисления $\sin(x)$ в виде бесконечной суммы

$$\sin(x) = \sum_{i=1}^{\infty}(-1)^{i-1}\frac{x^{2i-1}}{(2i-1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - ...$$

для значения x, запрашиваемого с клавиатуры. Суммирование необходимо проводить до тех пор, пока модуль очередного слагаемого не станет меньше заранее заданного маленького числа $\epsilon = 10^{-4}$.

Вариант 24.

Составьте программу приближённого вычисления e^x в виде бесконечной суммы

$$e^x = \sum_{i=0}^{\infty}\frac{x^i}{i!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ...$$

для значения x, запрашиваемого с клавиатуры. Суммирование необходимо проводить до тех пор, пока модуль очередного слагаемого не станет меньше заранее заданного маленького числа $\epsilon = 10^{-3}$.

Вариант 25.

Составьте программу приближённого вычисления $\cos(x)$ в виде бесконечной суммы

$$\cos(x) = \sum_{i=1}^{\infty}(-1)^{i}\frac{x^{2i}}{(2i)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - ...$$

для значения x, запрашиваемого с клавиатуры. Суммирование необходимо проводить до тех пор, пока модуль очередного слагаемого не станет меньше заранее заданного маленького числа $\epsilon = 10^{-4}$.

Вариант 26.

Составьте программу приближённого вычисления $\mathrm{sh}(x)$ в виде бесконечной суммы

$$\mathrm{sh}(x) = \sum_{i=1}^{\infty} \frac{x^{2i-1}}{(2i-1)!} = x + \frac{x^3}{3!} + \frac{x^5}{5!} + ...$$

для значения x, запрашиваемого с клавиатуры. Суммирование необходимо проводить до тех пор, пока модуль очередного слагаемого не станет меньше заранее заданного маленького числа $\epsilon = 10^{-3}$.

Вариант 27.

Составьте программу приближённого вычисления $\mathrm{ch}(x)$ в виде бесконечной суммы

$$\mathrm{ch}(x) = \sum_{i=0}^{\infty} \frac{x^{2i}}{(2i)!} = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + ...$$

для значения x, запрашиваемого с клавиатуры. Суммирование необходимо проводить до тех пор, пока модуль очередного слагаемого не станет меньше заранее заданного маленького числа $\epsilon = 10^{-3}$.

Вариант 28.

Составьте программу приближённого вычисления $\sin(x)/x$ в виде бесконечной суммы

$$\frac{\sin(x)}{x} = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i+1)!} = 1 - \frac{x^2}{3!} + \frac{x^4}{5!} - ...$$

для значения x, запрашиваемого с клавиатуры. Суммирование необходимо проводить до тех пор, пока модуль очередного слагаемого не станет меньше заранее заданного маленького числа $\epsilon = 10^{-4}$.

3. ПОДПРОГРАММЫ. УКАЗАТЕЛИ

3.1. Понятия подпрограмм. Функции

В практике программирования часто имеется необходимость выполнять одни и те же вычисления, но при различных исходных данных. Для исключения повторений одинакового кода и упрощения программы повторяющиеся вычисления выделяют в самостоятельную часть программы, которая может быть использована многократно по мере необходимости. Такая автономная часть программы, реализующая определённый алгоритм, оформленная в виде отдельной синтаксической конструкции, снабжённая именем и допускающая обращение к ней из различных частей программы, называется подпрограммой. Другими словами подпрограмма — это последовательность операторов, которые определены и записаны только в одном месте программы, однако их можно вызвать для выполнения из одной или нескольких точек программы. Каждая подпрограмма определяется уникальным именем.

Активное использование подпрограмм обусловлено также тем обстоятельством, что алгоритм решения задачи часто проектируется путем декомпозиции всей задачи на отдельные подзадачи. Обычно подзадачи оформляются как подпрограммы. В языке C++ подпрограммы реализуются в виде функций. Функция — это именованная последовательность описаний и операторов. Функция может располагаться как в файле основной программы, так и в отдельном файле. По определению функция должна возвращать вызывающей её программе результат свой работы. Однако, если по смыслу задачи возвращения результата не требуется, то при описании функции используют служебное слово `void` для типа

возвращаемого результата.

3.1.1. Описание функций

Описание функции выполняется согласно следующему шаблону

```
Тип ИмяФункции (список параметров)
{
  Тело функции;
}  //.
```

`Тип` определяет тип результата, возвращаемого функцией. Если тип не указан, то считается, что функция возвращает результат типа `int`.

Параметры функции — это переменные, посредством которых передаются данные из места вызова функции в её тело. Список параметров состоит из перечня их типов и имён, перечисленных через запятую. Если функция не имеет параметров, то их список может отсутствовать, но круглые скобки всё равно необходимы. Перечислим несколько примеров заголовков функций:

```
float diskr(float a, float b, float c)  //,
void podprogramma()  //,
int prog1()  //.
```

Передача результата (значения) из функции в вызывающую программу осуществляется с помощью оператора `return`. Использование этого оператора весьма многообразно. Если после оператора `return` находится возвращаемое значение, то выполнение функции прекращается и значение передаётся вызывающей программе. Если `return` используется без возвращаемого значения, то его назначение состоит в том, чтобы прекратить выполнение функции. Функция

может содержать один, несколько или ни одного оператора `return`. В последнем случае выполнение функции прекращается после выполнения последнего её оператора.

3.1.2. Область действия переменных

Область действия (видимости) переменной — это правила, которые устанавливают, какие данные доступны из данного места программы. В языке С каждая функция — это отдельный блок программы. Попасть в тело функции нельзя иначе, как через вызов данной функции.

С точки зрения области действия переменных различают три вида переменных: глобальные, локальные и формальные параметры. Правила области действия определяют, где каждая из них может применяться.

Локальные переменные — это переменные, объявленные внутри блока, в частности внутри функции. Язык С поддерживает простое правило: переменная может быть объявлена внутри любого блока программы. Область действия локальной переменной — блок. Локальная переменная существует пока выполняется блок, в котором эта переменная объявлена. При выходе из блока эта переменная (и её значение) теряется.

Формальные параметры — это переменные, объявленные при описании функции как её аргументы. Функции могут иметь некоторое количество параметров, которые используются при вызове функции для передачи значений аргументов. Формальные параметры могут использоваться в теле функции так же, как локальные переменные, которыми они по сути дела и являются. Область действия формальных параметров — блок, являющийся телом функции.

Глобальные переменные — это переменные, объявленные вне какой-либо функции. В отличие от локальных гло-

бальные переменные могут быть использованы в любом месте программы ниже их объявления. Область действия глобальной переменной — вся программа.

Использование глобальных переменных имеет свои недостатки: они занимают память в течение всего времени работы программы, использование глобальных переменных делает функции менее общими и затрудняет их использование в других программах.

3.2. Указатели

Понимание и правильное использование указателей является основой для создания профессиональных программ на языке С. Указатель — это переменная, которая предназначена для хранения и использования в программе адреса некоторого объекта. Здесь имеется в виду адрес в памяти компьютера. Адрес представляет собой простое целое число, но его нельзя трактовать как переменную или константу целого типа. Если переменная по смыслу является указателем, то она должна быть соответствующим образом объявлена. Форма объявления указателя следующая:

```
Тип *Имя_переменной;  //.
```

В этом объявлении тип — некоторый допустимый для языка С (С++) тип данных. Это тип той переменной, адрес которой будет хранить указатель. Знак * означает, что следующая за ним переменная является указателем. Например:

```
char *ch;
int *temp, i, *j;
float *pf, f;   //.
```

Здесь объявлены указатели ch,temp,j,pf, переменная i типа int и переменная f типа float.

С указателями связаны две специальные операции: & и *. Обе эти операции являются унарными, т. е. имеют один операнд, перед которым они ставятся. Операция & соответствует по смыслу операции взятия (определения) адреса. Операция * является операцией взятия (определения) значения по указанному адресу. Данные операции нельзя спутать с соответсвующими им по написанию бинарными операциями, поразрядным AND и операцией умножения, так как они являются унарными, что всегда видно из контекста программы.

При объявлении указателя очень важным является базовый тип, который сообщает компилятору сколько байт памяти занимает переменная, на которую указывает данный указатель. Простейшие действия с указателями проиллюстрируем на примере следующей программы:

```
# include <iostream.h>
int main (void)
{
//Объявляем переменные x и y
   float x= 12.3, y;
// Объявляем указатель p
   float *p;
// Присваиваем указателю p адрес переменной x
   p=&x;
// Переменной y присваиваем значение
// переменной, адрес которой
// содержит указатель p
   y=*p;
// Выводим на экран x и y
   cout << "x=" << x << "  y=" << y << endl;
```

```
// Увеличиваем на 1 значение,
// взятое по указателю p
   (*p)++;
// Выводим на экран x и y
   cout << "x=" << x << "  y=" << y << endl;
// Добавляем 1 к произведению значения,
// взятого по указателю p на y
   y=1+(*p)*y;
// Выводим на экран x и y
   cout << "x=" << x << "  y=" << y << endl;
} // конец программы.
```

К указателям можно применять операцию присваивания, если они являются указателями одного типа, например:

```
# include <iostream.h>
int main(void)
{int x=12;
int *p, *g;
p=&x;
g=p;
printf("%p\n",p);
printf("%p\n",g);
printf("%d\n%d\n",x,*g);
} // конец программы.
```

В этом примере приведена спецификация формата `%p` функции `printf()`, которая используется для вывода адреса памяти в шестнадцатеричной форме.

Нельзя создать переменную типа `void`, но можно создать указатель на такой тип. Указателю на `void` можно присвоить значение указателя любого другого типа. При обратном присваивании необходимо использовать явное преоб-

разование указателя на `void`. Например, рассмотрим следующий фрагмент:

```
void *pv;
float f, *pf;
pf=&f;
pv=pf;
pf=(float*)pv; // .
```

Здесь указатель `pv` приводится к типу `(float*)`, то есть к указателю на переменную типа `float`.

В языке С допустимо присвоить указателю любой адрес памяти. Однако если объявлен указатель на целое (`int *p`), а по адресу, который присвоен данному указателю, находится переменная `x` другого типа, то при компиляции программы будет выдано сообщение об ошибке в строке `p=&x`. Эту ошибку можно исправить, преобразовав адрес переменной `x` к нужному типу указателя явным преобразованием типа: `p=(int*)&x;` но при этом теряется информация о типе переменной `x`.

Над указателями можно производить арифметические операции: сложение и вычитание. Арифметические действия над указателями имеют свои особенности. Рассмотрим программу:

```
#include <stdio.h>
int main(void)
{
int *p;
int x=12;
p=&x;
printf("%p\n%p",p,++p);
}  //.
```

При выполнении этой программы увидим, что в результате операции ++p значение указателя p увеличиться на 2, а не на 1. Это правильно, так как следующее значение указателя указывает на адрес следующего целого, а не на следующий адрес (целое занимает 2 байта). Таким образом, при каждой операции ++p значение указателя будет увеличиваться на количество байт, занимаемых переменной базового типа указателя.

К указателям можно прибавлять или вычитать некоторое целое. Пусть указатель `p` имеет значение 4000 и указывает на целое. Тогда в результате выполнения оператора `p=p+5;` значение указателя станет равным 4010. Общая формула для вычисления значения указателя после выполнения операции `p=p+n;` будет иметь вид:

`p=p+n*Размер_в_байтах_переменной_базового_типа.`

Аналогичны правила вычитания целых констант из значения указателя. Можно также вычитать один указатель из другого. Другие арифметические операции над указателями запрещены, т. е. нельзя складывать два указателя, умножать друг на друга, делить и т. д.

Указатели можно сравнивать. Применимы все 6 операций сравнения. Сравнение p<g дает "истину если адрес, находящийся в p, меньше адреса, находящегося в g.

В языке С допускается использование указателей, указывающих на указатель [4]. «В этом случае описание имеет следующий вид:

```
int **point;  //.
```

Здесь `point` имеет тип указатель на указатель `int`. Соответственно, чтобы получить целочисленное значение переменной, на которую указывает `point`, надо в выражении использовать `**point`. Рассмотрим пример:

```
#include <stdio.h>
int main(void)
{
int i;
int *pi;
int **ppi;
i=12;
pi=&i;
ppi=&pi;
printf("i = %d pi = %p ppi = %p
**ppi = %d\n",i,pi,ppi,**ppi);
}  //.
```

После того как указатель был объявлен, но до того как ему было присвоено значение, указатель содержит неопределённое значение. Попытка использовать такое значение может вызвать ошибку при выполнении программы и даже нарушить работу операционной системы.» Всем указателям, не используемым в данный момент целесообразно присваивать предопределённое значение NULL. Оно является в некотором смысле «заземлением» указателя.

3.3. Передача параметра по ссылке

В языке С все аргументы функции передаются по значению. Это значит, что при вызове функции в системный стек помещаются фактические значения аргументов функции. В функции создаются локальные копии аргументов, значения которых считываются из стека. Далее функция использует эти локальные переменные и может изменять их. При выходе из функции локальные переменные уничтожаются, изменённые значения параметров теряются. Таким образом, в языке С вызванная функция не может изме-

нить значения переменных, указанных в качестве фактических параметров при обращении к ней. Например, функция `swap()`, которая должна менять значения параметров местами, не будет этого делать:

```
void swap(int a, int b)
{
int tmp=a; a=b; b=tmp;
} //.
```

Но тем не менее функцию можно приспособить для изменения аргументов. Для этого необходимо в качестве параметра передавать адрес переменной, которую надо изменять, т. е. передавать указатель на переменную. Такой приём в языке C называется передачей параметра через указатель. Вызванная функция должна описывать аргумент как указатель и обращаться к фактической переменной косвенно, через разыменование указателя. Теперь функцию `swap()` можно описать следующим образом:

```
void swap(int *a, int *b)
{
int tmp=*a; *a = *b; *b=tmp;
} //.
```

Тогда для обмена местами значений переменных `x` и `y` необходимо использовать функцию `swap(&x,&y)`.

Ещё одним способом косвенной передачи параметра функции является передача по ссылке. Тогда аргументами функции `swap()` будут «ссылки» на другие переменные

```
void swap(int &a, int &b)
{
int tmp=a; a = b; b=tmp;
} //.
```

При обращении к функции в качестве фактических параметров необходимо указывать имена переменных `x` и `y`: `swap(x,y)`. Подробнее о ссылках можно почитать в [2].

3.4. Динамическое выделение памяти

Помимо рассмотренных элементарных действий с указателями с их помощью можно также создавать новые переменнные и выделять память динамически в ходе работы программы. Для этого предназначен оператор `new`:

```
указатель = new тип;
```

например,

```
int *p; // указатель на целое
p = new int; // выделяем память.
```

После операции `new` указывается тип данных, которые будут храниться в выделяемой области памяти, адрес начала выделенной области присваивается указателю. После этого можно пользоваться выделенными ячейками, например

```
*p = 155; // присвоили значение.
```

Когда динамически выделенная память больше не нужна её можно освободить с помощью оператора delete:

```
delete указатель;
```

например,

```
delete p; // освободили память.
```

Удаления указателя при этом не происходит, память, адрес которой находится в указателе, объявляется свободной.

3.5. Рекурсивные функции

Рекурсивная функция — это функция, в теле которой имеется вызов самой себя. Использование рекурсивных функций бывает удобным при программировании ряда задач, например вычислении факториала некоторого числа N или быстрой сортировки массива.

```
int factorial(int n)
{
int a;
if (n==1) return 1;  // предусмотрели выход
a = factorial(n-l)*n; // вызов себя
return a;
}  //.
```

Вызов функции при рекурсии не создаёт новую копию функции в памяти, а создаёт новые копии локальных переменных и параметров. Из рекурсивной функции необходимо предусмотреть выход, чтобы не спровоцировать «зависание» вычислительной системы. Необходимо помнить, что при большом числе рекурсивных вызовов будет происходить быстрое заполнение стека, размер которого ограничен. Это может вызвать остановку программы. Поэтому, использование рекурсии как метода программирования, должно быть осторожным.

4. ОСОБЕННОСТИ МУЛЬТИФАЙЛОВОГО ПРОГРАММИРОВАНИЯ

4.1. Компиляция и компоновка

Как правило текст серьёзной программы неудобно размещать в одном файле. В такой программе неудобно разбираться, её трудно модифицировать. Целесообразно разбивать текст программы на отдельные функционально законченные блоки и размещать их в разных файлах. Затем файлы с исходными кодами блоков компилируются отдельно, и только потом полученные объектные коды компонуются в единый исполняемый файл. Рассмотрим этот процесс подробнее. Создадим каталог проекта. Разделим текст программы вычисления дохода по вкладу, приведённой на стр. 34 на три файла. В первом из них `func.cpp` разместим функцию вычисления дохода

```
float fun_dohod(float sum, float srok, float stavka)
{
return (sum*stavka/100)*srok/365;
} //.
```

Второй файл `func.h` — заголовочный, в нем находится прототип функции `fun_dohod`.

```
float fun_dohod(float,float,float); //.
```

Прототипом функции называется её заголовочная строка с указанием имени функции и типов аргументов и результата. Третий файл main.cpp содержит главную функцию `main`, в которой используется функция `fun_dohod` для вычисления дохода по вкладу

```
#include <iostream.h>
#include "func.h"
float sum,      //сумма вклада
      srok,     //срок вклада
      stavka,   //процентная ставка
      dohod;     //доход по вкладу
int main(void)
{
cout << "Вычисление дохода по вкладу\n";
cout << "Введите исходные данные:\n";
cout << "Величина вклада (руб) ->";
cin >> sum;
cout << "\n";
cout << "Срок вклада (дней) ->";
cin >> srok;
cout << "\n";
cout << "Процентная ставка ->";
cin >> stavka;
cout << "\n";
dohod=fun_dohod(sum, stavka, srok);
sum=sum+dohod;
cout << "\n";
cout << "-------------------------\n";
cout << "Доход:" << dohod << "руб\n";
cout << "Конечная сумма:" << sum << "руб\n";
} //.
```

Заголовочный файл используется для «подключения» функции `fun_dohod` из файла func.cpp к главной программе с помощью строки `#include "func.h"`. Для иллюстрации всего процесса получения исполняемого файла выполним компиляцию двух файлов с исходным текстом программы командами

```
g++ -c func.cpp и
g++ -c main.cpp.
```

В результате в каталоге проекта появятся два файла с объектным кодом `func.o` и `main.o`. Заглянуть внутрь объектного кода помогает утилита `nm`. Команда

```
nm func.o
```

выводит на экран так называемый список символов программы, в котором упоминается имя функции `fun_dohod`.

```
...
00000000 T __Z9fun_dohodfff
```

Заметим, что напротив всех символов файла func.o находятся нули. Это означает, что символам не поставлен в соответствие реальный адрес или смещение относительно некоторого адреса. Команда

```
nm main.o
```

выводит список символов объектного кода `main.o`.

```
...
         U __Z9fun_dohodfff
...
000000fe T _main
00000004 B _srok
00000008 B _stavka
00000000 B _sum
```

Здесь для функции `main` указано значение 000000fe, а напротив строки с именем функции `fun_dohod` адрес не указан, что свидетельствует о необходимости связывания имён

с помощью таблиц символов. В процессе компоновки высчитываются адреса, соответствующие именам списка символов и подставляются в нужные места таблицы, а также подключаются функции из используемых стандартных библиотек. Вывод команды

```
nm a.exe
```

черезвычайно длинный, однако в нем присутствуют строки

```
...
00401640 T __Z9fun_dohodfff
...
004013ee T _main
...
```

сопоставляющие описанные в программе функции с реальными адресами или смещениями адресов.

4.2. Автоматическая сборка проекта

Согласно предыдущему пункту для создания исполняемого файла из двух файлов с исходным текстом программы потребовалось вводить три команды: две для компиляции каждого файла и одну для компоновки. Предположим теперь, что файлов с исходным текстом довольно много, причём функции, описанные в одних используют функции, описанные в других файлах. Тогда изменение текта программы при отладке приведёт к необходимости перекомпиляции многих файлов. Для автоматизации процесса компиляции существует утилита **make**. Её работа происходит по заранее подготовленному сценарию, расположенному в специальном файле `Makefile` в каталоге проекта. Содержимое `Makefile`

состоит из элементов трёх типов: комментариев, макроопределений и целевых связок. Комментарием считается строка, начинающаяся знаком диез. Макроопределения позволяют назначить имя какой-либо строке, а затем подставить эту строку в необходимом месте. Целевые связки являются обязательным содержимым файла сценария и в свою очередь состоят тоже из трёх элементов: цель, зависимости и правила. В связках указывается, что нужно сделать для компиляции, что для этого нужно и как (с помощью какой команды) это сделать.

Цель — это идентификатор (какое-либо имя или «волшебное слово») с двоиточием. Зависимости — это список файлов или других целей, разделённых пробелом, необходимых для успешного осуществления цели. Правила — команды, которые необходимо выполнить для достижения цели. Для рассмотренного выше двухфайлового проекта можно использовать следующий Makefile.

```
# Пример файла сценария Makefile

prog:  main.o func.o
        g++ -o prog.exe main.o func.o

main.o: main.cpp
        g++ -c main.cpp

func.o: func.cpp
        g++ -c func.cpp

clean:
        rm -f  *.o prog.exe
```

Здесь описаны четыре цели: `prog`, `main.o`, `func.o`, `clean`.

Первая цель `prog` служит для компоновки объектных кодов и получения исполняемого файла `prog.exe`. Она зависит от наличия файлов с объектными кодами `main.o` и `func.o`, которые в свою очередь являются целями. Для достижения цели prog используется правило (команда)

```
g++ -o prog.exe main.o func.o.
```

Все правила обязательно начинаются с новой строки и с символа табуляции. Цели `main.o` и `func.o` оформлены одинаково. Их зависимостями служат файлы с исходными текстами программы, а правила осуществляют компиляцию с подавлением компоновки. Наконец, последняя цель `clean` служит для очистки проекта от результатов компиляции. Она не имеет зависимостей и её правило представляет собой команду удаления файлов с расширением `o` и исполняемого файла `prog.exe`. Команда удаления заимствована из Linux. Реализация утилиты `make` для Windows пользуется встроенным эмулятором терминала Linux, поэтому тоже использует команду удаления `rm`.

После написания `Makefile` для сборки проекта достаточно запустить утилиту `make` и указать ей цель, например

```
make prog
```

или

```
make clean.
```

Запуск `make` без параметров приводит к сборке первой цели, которая по умолчанию считается главной.

Современные интегрированные среды программирования содержат также утилиты, котрые анализируют исходный текст программы, зависимости между файлами и сами составляют `Makefile`. Именно поэтому в пунктах 1.5.1 и 1.5.2 упоминалось о создании целей и файла сценария `Makefile`.

5. ЛАБОРАТОРНАЯ РАБОТА №2 РАБОТА С МАССИВАМИ

Цели работы:

- овладение практическими навыками работы с массивами в C++;
- освоение методов сортировки массивов.

5.1. Одномерные массивы

Массив — это формальное объединение нескольких однотипных переменных (числовых, символьных, строковых и т.п.), рассматриваемое как единое целое. Описать массив можно либо при описании переменных, либо с использованием описания нового типа. При описании массива необходимо указать: имя массива, тип его элементов, количество элементов в массиве.

Описание массива в программе выглядит так:

```
Тип_Элементов_Массива   Имя_Массива[Количество];
```

например,

```
int a[100],b[100],c[100],d[100];
```

Здесь определено 4 массива с именами a,b,c и d. Все они состоят из элементов целого типа, пронумерованных целыми числами от 0 до 99. Элементы массива *всегда* нумеруются целыми числами, начиная с 0. При описании элементам массива можно присвоить значения:

```
int x[10] = {20, 2, 5, 89, 7, 5, 3, 6, 1, 4};
```

5.1.1. Действия над элементами массивов

Доступ к элементам массива осуществляется путем указания имени переменной массива, за которым в квадратных скобках помещается значение индекса (порядкового номера) элемента. Примеры задания индекса: M[5] – непосредственно числом; M[k] – косвенно через переменную k; M[k1+5] – косвенно через выражение k1+5; M[abs(i)] – косвенно через значение функции.

Например, с помощью оператора

```
mas[2]=34;
```

элементу массива `mas` с индексом 2 присваивается значение 34. Оператор

```
cout << mas[5] << endl;
```

выведет на экран значение элемента массива mas с номером 5.

Всем элементам одного массива *нельзя* присвоить значения элементов другого массива с помощью одного оператора присваивания. Так, если заданы следующие массивы:

```
int x[10], y[10];
```

то *недопусти́м* следующий оператор присваивания:

```
x=y; //.
```

В языке С над массивами не определены операции отношения. Сравнивать массивы можно только поэлементно. К отдельным элементам массива можно применять стандартные процедуры и функции, предусмотренные в языке. Перечень допустимых стандартных подпрограмм зависит от типа элементов массива.

Используя в качестве индекса переменную цикла `for` можно обратиться к каждому элементу массива по очереди. Так с помощью фрагмента программы

```
for (i=1;i<=25;i++) mas[i]=0;
```

всем элементам массива присваивается нулевое значение.

Использование массивов вместо одиночных переменных даёт преимущество при обработке многих однотипных данных. Пусть, например, метеорологическая станция производит измерение температуры воздуха ежедневно в один и тот же час. Требуется найти среднемесячную температуру. Использование массивов позволяет объединить данные измерений в один массив, а также компактно запрограммировать их обработку.

Пример.

Дана последовательность температур t0 ... t30. Организовать массив для хранения этой последовательности. Определить среднемесячную температуру.

```
#include <iostream.h>
int main(void)
{
 float t[31];           //описание массива
 int i;                 //параметр цикла for
 float s;               //сумма элементов
 for (i=0;i<=30;i++)    //заполнение массива
   {
   cout << "введите элемент с номером" << i;
   cin >> t[i];
   };
s=0;     //обнуление суммы
```

```
for (i=0;i<=30;i++) s=s+t[i]; //вычисление суммы
cout << "среднемесячная температура" << s/31;
} // конец программы.
```

5.2. Связь указателей и массивов

Существует тесная связь между массивами и указателями. Она заключается в том, что в объявленном массиве его имя является указателем на массив, а точнее, на первый элемент массива. Таким образом, если был объявлен массив `int t[31];` то `t` является указателем на массив, а операторы `pl=t;` и `pl=&t[0];` приведут к одному и тому же результату. Для того чтобы получить значение 8-го элемента массива `t` можно написать `t[7]` или `*(t+7)`. Результат будет один и тот же. Преимущество второго варианта заключается в том, что арифметические операции над указателями выполняются быстрее, чем действия с элементами массива.

В качестве примера перепишем предыдущую программу вычисления среднемесячной температуры с использованием указателей

```
#include <iostream.h>
int main(void)
{
 float t[31];           //описание массива
 int i;                 //параметр цикла for
 float s;               //сумма элементов
for (i=0;i<=30;i++)    //заполнение массива
   {
   cout << "введите элемент с номером" << i;
   cin >> *(t+i);
   };
s=0;     //обнуление суммы
```

```
for (i=0;i<=30;i++) s=s+*(t+i); //вычисление суммы
cout << "среднемесячная температура" << s/31;
} // конец программы.
```

5.3. Заполнение массивов

Работа с массивами требует их предварительного заполнеия, то есть присваивания начального значения каждому элементу массива. Для заполнения массива можно, например, использовать ввод с клавиатуры:

```
cout << "Введите значения для элементов массива:\n";
for (i=0;i<=99;i++)
  {
  cout << "x1["  << i << "]=";
  cin >> x1[i];
  };  //.
```

Однако при отладке программы приходится вводить с клавиатуры много элементов массива. Чтобы избавиться от этой утомительной работы целесообразно заполнять массивы случайными числами. Для этого каждому элементу массива присваивается значение с помощью датчика псевдослучайных чисел `rand()`.

Функция `rand()` определена в библиотеке `stdlib` и умеет генерировать псевдослучайное целое число между 0 и константой `RAND_MAX`, определённой в той же библиотеке. Перед использованием датчик случайных чисел необходимо инициализировать начальным значением. Для этого предназначена функция `srand(unsigned long int)`, которая получает целый положительный аргумент и задаёт начальное число для функции `rand()`. Агрумент функции `srand()` можно вводить с клавиатуры. Тогда при вводе одинаковых

значений можно получить одинаковые последовательности случайных чисел, что полезно для отладки программы. При вводе разных значений аргумента функции `srand()` генерируемые последовательности случайных чисел будут разными. Если вводить с клавиатуры начальное число нежелательно, то можно считывать его с системного таймера, например, так `srand(time(NULL))`. Функция `time()` принадлежит библиотеке `time`. Таким образом, для заполнения массива целыми числами от 0 до N можно использовать фрагмент программы

```
#include<stdlib.h>
#include<time.h>
...
srand(time(NULL));       //Инициализируем датчик
for (i=0;i<=99;i++)      //Для каждого элемента
  x1[i]=rand()%(N+1);    //Присваиваем случайное
```

Здесь использована операция взятия остатка от деления для приведения диапазона значений случайной величины к отрезку $[0, N]$. Если необходимо заполнить массив вещественными псевдослучайными числами из отрезка $[a, b]$, можно писать так:

```
x1[i]=a+(b-a)*1.0*rand()/RAND_MAX; //.
```

Здесь результат деления `rand()/RAND_MAX` по смыслу является вещественным случайным числом из отрезка $[0, 1]$. Однако компилятор C (C++) при делении целого на целое не выполняет приведения типа результата к вещественному числу. Поэтому в выражение введено умножение на вещественную единицу 1.0. Поскольку в правой части выражения встретился хоть один вещственный множитель, результат будет приведён к вещественному типу. Преобразование

$y = a + (b - a)x$ позволяет трансформировать диапазон значений $x \in [0, 1]$ в диапазон значений $y \in [a, b]$.

5.4. Поиск элементов, удовлетворяющих заданному условию

Зачастую приходится выполнять поиск в неупорядоченном массиве элементов, которые удовлетворяют заданному условию, например, условию положительности. Следующая программа заполняет массив целыми случайными числами от -100 до 100, определяет количество положительных элементов и выдаёт сообщение на экран.

```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>
int main(void)
{
int  m[30];   //Описание массива
int i;        //параметр цикла for
int n;        //счётчик положительных элементов
srand(time(NULL));     //Инициализация датчика
for (i=0; i<=29;i++)   //заполнение массива
  m[i]=rand()%201-100;
n=0;                  //обнуление счётчика элементов
for (i=0; i<=29;i++)
  if (m[i]>0)            //Условие положительности
      n++;                //Наращиваем на 1 счетчик
cout << ‘‘в массиве ‘‘ << n << ‘‘ положительных эл-ов’’;
return 0;
} // конец программы.
```

Для подсчёта количества элементов, удовлетворяющих условию положительности введена переменная n с нулевым на-

чальным значением. Далее циклически выполняется поочерёдное сравнение каждого элемента массива с нулем. В случае положительности очередного элемента переменная n наращивается на единицу.

5.5. Определение максимального элемента и его положения в массиве

Рассмотрим пример программы, которая заполняет вещественный массив случайными числами и определяет значение и индекс (номер) максимального элемента этого массива

```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>
int main(void)
{
float  m[30];  //Описание массива
int i;         //параметр цикла for
float  max;    //значение максимального элемента
int  t;        // индекс (номер) макс. элемента

srand(time(NULL));  //Инициализация датчика
for (i=0; i<=29;i++)   //заполнение массива
  m[i]=10.0*rand()/RAND_MAX;
// допустим, что 1-й элемент максимален
max=m[0];
t=0;
for (i=1; i<=29;i++) //все остальные элементы
  if (m[i]>max)      //сравниваем с максимальным
              {
              max=m[i];
```

```
                t=i;
                };
cout << "максимальный элемент = " << max;
cout << "номер максимального элемента " << t;
return 0;
} // конец программы.
```

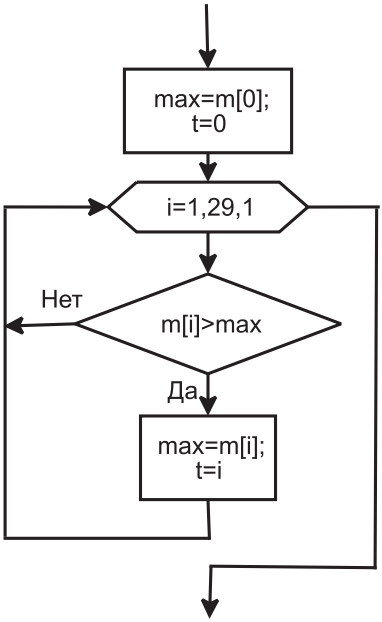

Рис. 16. Блок-схема алгоритма поиска максимального элемента массива

Алгоритм поиска максимального элемента достаточно прост. Для хранения значения максимального элемента предусмотрим переменную `max` того же типа, что и элементы массива, а для хранения индекса максимального элемента — переменную целого типа `t`. Предположим сначала, что

первый элемент массива (с номером 0) является максимальным: `max=m[0]; t=0;`. Далее поочерёдно сравниваем каждый элемент массива с текущим значением переменной `max`. Если очередной элемент превысил максимальное значение, то присваиваем переменной `max` значение этого элемента, а переменной `t` — значение его номера. На рис. 16 показан фрагмент блок-схемы описанного алгоритма. Если массив содержит несколько элементов с одинаковым максимальным значением, то приведённая программа найдёт только первый из них. Аналогично выполняется поиск минимального элемента массива.

5.6. Упорядочивание (сортировка) массивов

В большинстве случаев массивы применяются для хранения большого количества однотипных данных, создания и организации баз и банков данных и т.д. Во многих случаях номер (индекс) элемента в массиве не играет большой роли. Важнó само значение элемента, т.е. сама информация. Поиск нужной информации в неупорядоченном массиве или банке данных возможен только способом последовательного перебора всех элементов. Существуют алгоритмы быстрого поиска информации, но все они работают только с упорядоченными массивами данных. Поэтому задача сортировки элементов массивов является очень актуальной.

Пусть дан числовой массив x из n попарно различных элементов. Требуется переставить элементы массива так, чтобы они были упорядочены по возрастанию

$$x[0] < x[1] < ... < x[n-1].$$

Существует много алгоритмов решения этой задачи.

Алгоритм сортировки перебором.

Очевидно, что первое место в массиве должен занять минимальный элемент массива, второе — наименьший из всех остальных, третье — наименьший из оставшихся и т.д. Следосательно, для сортировки необходимо выполнить следующую последовательность действий:

1) определить минимальный элемент массива;
2) поменять его местами с первым элементом;
3) определить минимальный элемент среди оставшихся;
4) поменять его местами со вторым элементом и т.д.

Эта последовательность действий должна выполняться до тех пор, пока не будет определён последний минимальный элемент.

Всю операцию по упорядочиванию массива можно разбить на более простые задачи. Первая — поиск минимального элемента среди элементов с номерами i..n. Величина i должна быть равна сначала 1, затем 2, 3 и т.д до n-1. Вторая — замена местами минимального элемента с номером t и элемента номером i. Поменять местами две переменные x и y можно двумя способами:

1) с использованием вспомогательной переменной v=x; x=y; y=v;
2) без использования вспомогательной переменной x=x+y; y=x-y; x=x-y; но только для числовых данных.

На рис. 17 изображена блок-схема алгоритма сортировки массива методом перебора.

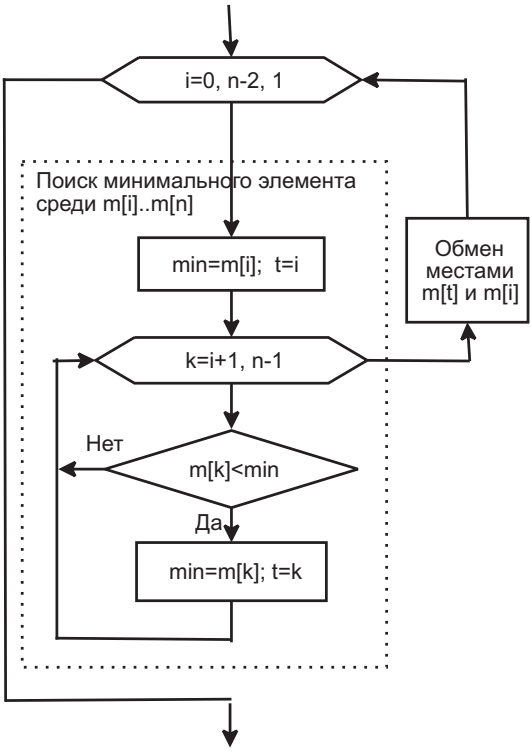

Рис. 17. Блок-схема алгоритма сортировки массива методом перебора

Алгоритм сортировки методом пузырька.

Данный способ сортировки основан на последовательном сравнении соседних элементов массива. Если два соседних элемента расположены не в нужной последовательности, то меняем их местами. Так повторяем до тех пор, пока в очередном проходе не сделаем ни одного обмена, т.е. массив будет упорядоченным. На рис. 18 показана блок-схема алгоритма сортировки по возрастанию методом пузырька (m - массив для сортировки с начальным индексом 0, n - размер

массива).

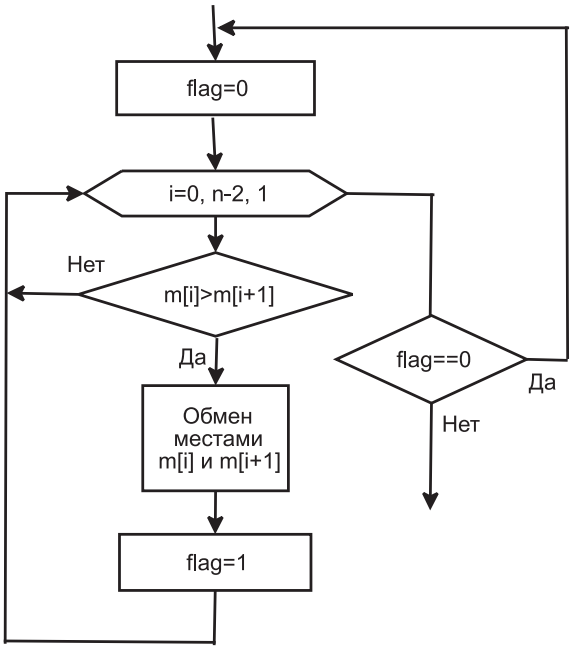

Рис. 18. Блок-схема алгоритма сортировки массива методом пузырька

В качестве примера отсортируем по возрастанию массив

```
5 2 7 4 3 1 8 6.
```

Первый проход по массиву приводит к обмену местами элементов 5 и 2, 7 и 4, 7 и 3, 7 и 1, 8 и 6:

```
2 5 7 4 3 1 8 6,
2 5 4 7 3 1 8 6,
2 5 4 3 7 1 8 6,
```

```
2 5 4 3 1 7 8 6,
2 5 4 3 1 7 6 8.
```

В результате последнее место занял максимальный элемент 8. В течение второго прохода меняются местами 5 и 4, 5 и 3, 5 и 1, 7 и 6:

```
2 4 5 3 1 7 6 8,
2 4 3 5 1 7 6 8,
2 4 3 1 5 7 6 8,
2 4 3 1 5 6 7 8.
```

Третий проход приводит к обмену 4 и 3, 4 и 1:

```
2 3 4 1 5 6 7 8,
2 3 1 4 5 6 7 8,
```

четвёртый проход — 3 и 1:

```
2 3 4 1 5 6 7 8,
2 1 3 4 5 6 7 8,
```

и, наконец, пятый проход — 2 и 1:

```
1 2 3 4 5 6 7 8.
```

В результате для сортировки этого массива потребовалось 5 проходов. В общем случае количество необходимых проходов зависит от степени «отсортированности» исходного массива, но не превышает $n - 1$.

Алгоритмы быстрой сортировки.

Метод «быстрой» сортировки, предложенный К. Хоаром (C. Hoare), основан на разделении массива на два непустых непересекающихся подмножества элементов. Для этого выбирается элемент массива (например, расположенный

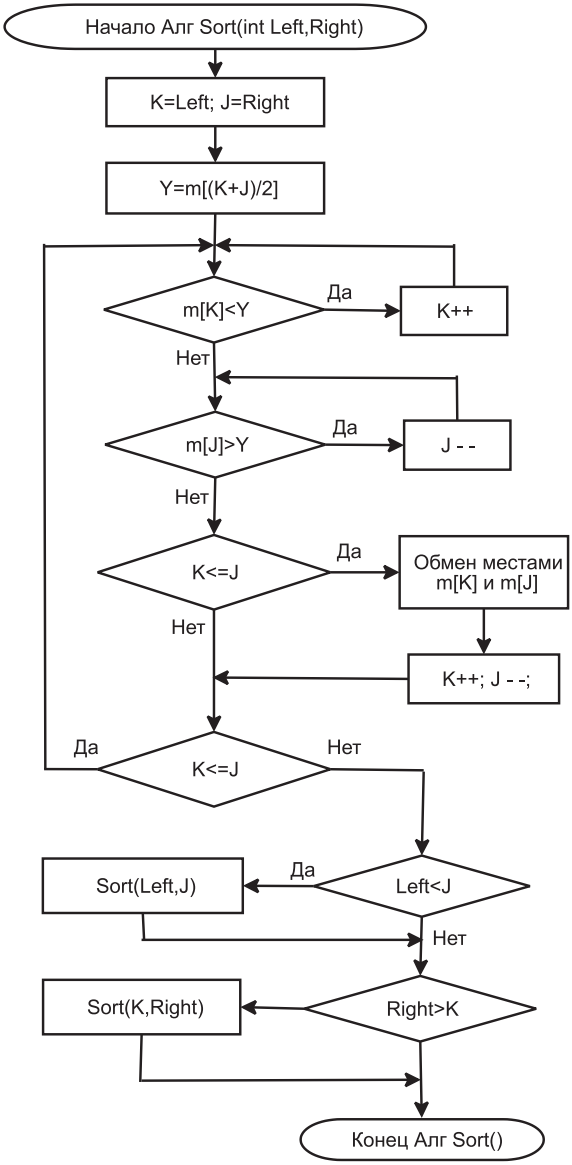

Рис. 19. Блок-схема алгоритма быстрой сортировки массива

посередине), запоминается его значение. Далее элементы переставляются так, чтобы в одной части массива (например вначале) оказались все элементы, значения которых не превосходят значения выбранного элемента, в в другой части — все элементы, не меньшие его. Тогда выбранный элемент окажется где-то посередине массива и будет служить границей раздела частей. При этом каждая из двух частей массива не является отсортированной. Аналогично следует поступить с каждой из полученных частей и так далее. На определенном этапе массив окажется полностью отсортированным. Как правило использование «быстрой» сортировки даёт лучшие результаты по скорости по сравнению со всеми остальными методами.

Поскольку алгоритм разделения на части одинаков как для всего массива, так и для его частей, целесообразно сделать его зависимым от аргументов — начального и конечного индекса сортируемой части массива. Тогда один и тот же алгоритм можно применить как для упорядочивания всего массива, так и для его частей. Следовательно «быструю» сортировку удобнее всего формулировать в рекурсивном виде. Действительно, алгоритм сортировки всего массива после разделения его на части должен вызвать сортировку частей с указанием начального и конечного индексов каждой части.

Если известно, что массив не содержит повторяющихся элементов, то элемент, разделяющий массив на части можно выбирать произвольно. Блок-схема алгоритма быстрой сортировки массива с попарно различными элементами показана на рис. 19. Исходными данными для него являются величины Left — номер элемента, который считается левой границей сортируемой части массива и Right — номер элемента, который считается её правой границей. Сначала вво-

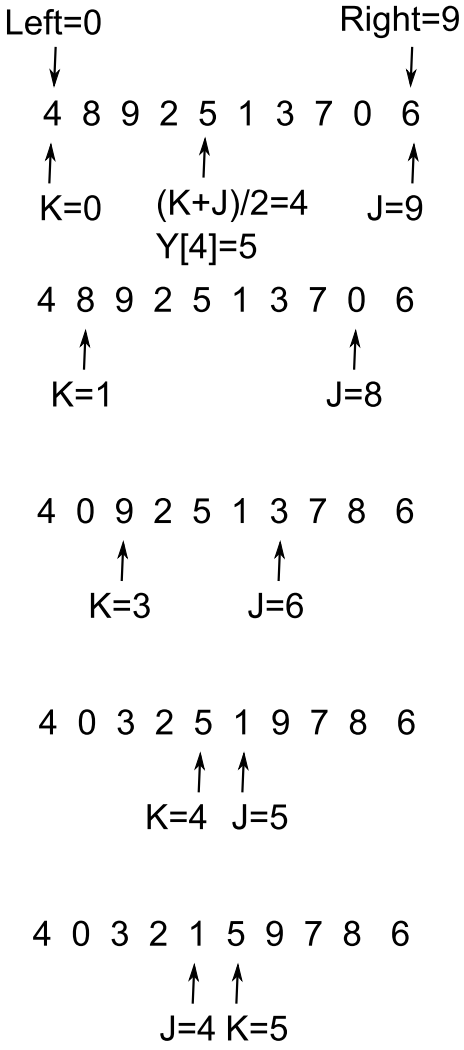

Рис. 20. Пример быстрой сортировки массива, шаг 1

дятся три локальные переменные: K с начальным значением левой границы, J с начальным значением правой границы и Y — значение элемента, расположенного посередине массива. Поскольку K и J являются переменными целого типа, то значение выражение $(K + J)/2$ по правилам языка C тоже приводится к целому типу. Так, например, для массива, изображенного на рис. 20, $Y = 5$.

Предположим, что необходимо упорядочить элементы массива по возрастанию. Будем увеличивать на 1 переменную K до тех пор, пока не найдётся элемент $m[K] \geq Y$. Переменную J будем уменьшать на 1 до тех пор, пока не найдётся элемент $m[J] \leq Y$. Для массива, показанного на рис. 20, это соответственно элементы m[1]=8 и m[8]=0. Если $K \leq J$, то меняем найденные элементы массива местами, увличиваем на 1 переменную K и уменьшаем на 1 переменную J. Если $K \leq J$, продолжаем увеличивать на 1 переменную K до тех пор, пока не найдется элемент $m[K] \geq Y$, а переменную J — уменьшать на 1 до тех пор, пока не найдется элемент $m[J] \leq Y$. Для массива, показанного на рис. 20, следующими найденными элементами являются m[3]=9 и m[6]=3. Меняем их местами и аналогично находим следующие элементы массива, подлежащие обмену: m[4]=5 и m[5]=1. Так происходит до тех пор, пока не выполнится условие $J \leq K$. Применительно к массиву рис. 20 J=4, K=5. Эти значения переменных J и K определяют границы деления массива на 2 части [Left, J] и [K, Right]. Далее алгоритм сортировки вызывается для каждой из этих частей отдельно.

На рис. 21 показан второй шаг — процесс сортировки левой и правой частей массива. В результате каждая из этих частей будет поделена ещё на две части (элементы с номерами 0,1,2; 3,4; 5,6; 7,8.9), для каждой из которых будет вызван алгоритм сотрировки.

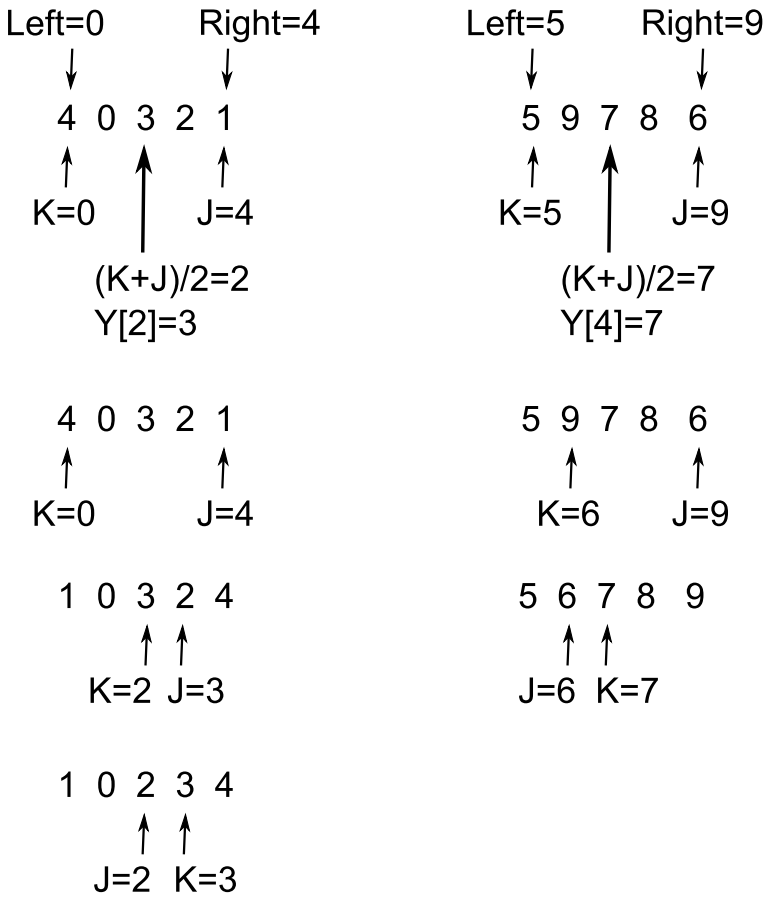

Рис. 21. Пример быстрой сортировки массива, шаг 2

Если исходный массив может содержать одинаковые элементы, то элемент, разделяющий массив на части уже нельзя выбирать произвольно. К примеру, массив (или его часть), состоящий из элементов 8 7 8 9 8, приведёт к зацикливанию алгоритма сортировки, так как крайние элементы и средний разделяющий элемент равны друг другу

m[Left]=m[Right]=Y. В этом случае вместо простого выбора $Y = m[(K + J)/2]$ необходимо применять более изощрённые способы выбора разделяющего элемента. Целесообразно выбираеть его близким к среднему арифметическому всех элементов массива и неравным первому и последнему элементам.

5.7. Удаление элементов из массива и вставка элементов в массив

До сих пор рассматривались массивы, размер которых известен заранее до компиляции программы. С помощью динамического выделения памяти довольно просто создать массив, размер которого неизвестен заранее. Пусть, например, размер массива n вводится с клавиатуры, затем выделяется память для всего массива:

```
int n;
cin >> n // ввод размера массива
...
int *massiv = new int[n]; // выделение памяти.
```

Здесь в одной строке объявляется указатель на целое massiv и выделяется память для n элементов массива. Казалось бы этот массив не имеет имени, однако указатель на его первый элемент `massiv` и есть имя массива. Следовательно доступ к элементам возможен как через разыменование указателя `*(massiv+k)`, так и через имя `massiv[k]`.

К сожалению, стандартный язык С не имеет средств для создания массивов переменной длины. В ходе выполнения программы длина массива меняться не может. Поэтому при необходимости работы с массивами переменной длины описывают массив заведомо бóльшего размера. Вместе с тем

описывают целую переменную, смысл которой — фактическая длина массива, то есть размер используемой его части. Хотя этот путь и нерационален, мы будем использовать его для работы с массивами переменной длины, чтобы изучить алгоритмы удаления и вставки элементов. Чтобы удалить элемент с номером g из массива mas, необходимо каждый элемент, начиная с g+1 и до последнего, переместить на место предыдущего элемента и уменьшить на 1 размер массива. Описанный алгоритм реализуется следующим фрагментом программы:

```
float mas[100];   //Описание массива
int i;            //параметр цикла for
int n;            //Фактический размер массива
.............................
for (i=g; i<=n-1; i++) mas[i]=mas[i+1];
n--;              //Уменьшение n на 1.
```

Для вставки элемента в массив на место с номером g необходимо предварительно освободить для него место. Для этого начиная с конца массива и до элемента с номером g каждый элемент перемещается на следующее за ним место, а размер массива увеличивается на 1. Затем присваивается новое значение элементу с номером g:

```
for (i=n-1; i>=g; i--) mas[i+1]=mas[i];
mas[g]=...
n++;              //Увеличение n на 1
```

5.8. Многомерные массивы

Элементы многомерных массивов нумеруются не одним, а несколькими (как минимум двумя) индексами. На-

пример, массив `float x2[10][20]` представляет собой двумерный массив, который можно интерпретировать как матрицу из 10 строк и 20 столбцов. Элементы этого массива могут принимать вещественные значения.

Массив `float x3[10][20][5]` представляет собой трёхмерный массив из 1000 элементов. Чаще всего в программах не используют массивов размерностью больше 2. Поскольку двумерные массивы по смыслу являются матрицами, им следует уделить больше внимания. Двумерный массив — структура данных, хранящая прямоугольную матрицу. В матрице каждый элемент определяется номером строки и номером столбца, на пересечении которых он расположен. В C++ двумерный массив представляется массивом, элементами которого являются одномерные массивы. Доступ к каждому отдельному элементу осуществляется обращением к имени массива с указанием индексов (первый индекс — номер строки, второй индекс — номер столбца). Все действия над элементами двумерного массива идентичны действиям над элементами одномерного массива. Только для инициализации двумерного массива используется вложенный цикл for, например

```
for (i=0; i<=9; i++)
  for (j=0; j<=19; j++)
    A[i][j] = 0;    //.
```

Для иллюстрации принципов работы с двумерными массивами разберём несколько примеров.

Пример 1.

Сформировать таблицу Пифагора (таблица умножения) и вывести её на экран.

```
#include <iostream.h>
```

```
#include <math.h>
int main(void)
{
int p[9][9];
int i, j;
for (i=0; i<=8; i++)        //Цикл по строкам
  for (j=0; j<=8; j++)      //Цикл по столбцам
    p[i][j]= (i+1)*(j+1);
for (i=0; i<=8; i++)
   {
   for (j=0; j<=8; j++) cout << p[i][j] << "\t";
   cout << endl;  //Перевод курсора на сл. строку
   }
}
```

Пример 2.

Задан двумерный массив b[10][10] целых чисел, заполненный случайными числами из отрезка [-10,10]. Найти и вывести на экран те элементы массива, которые больше заданного числа k.

```
#include <iostream.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
int main(void)
{
int  b[10][10];
int  i, j, k;
srand(time(NULL));
for (i=0; i<=9; i++)
  {
  for (j=0; j<=9; j++)
```

```
    {
      b [i][j]=rand()%21-10;
      cout << b[i][j] << " ";
    };
  cout << endl;
    };
cout << "Введите число k";
cin >> k;
for (i=0; i<=9; i++)
  for (j=0; j<=9; j++)
   if (b[i][j]>k) cout << b[i][j] << "\t";
cout << endl;
}  //.
```

Пример 3.

Задать и распечатать вещественный массив 10 на 10, состоящий из целых случайных чисел в интервале [1,100]. Найти сумму элементов, лежащих выше главной диагонали.

Главной является диагональ, проведённая из левого верхнего угла массива в правый нижний. При этом получается, что элементы, лежащие на главной диагонали будут иметь одинаковые индексы, а для элементов выше главной диагонали номер столбца будет всегда превышать номер строки.

```
#include <iostream.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
int main(void)
{
float A[10][10];
int i, k;
```

```
float S;
S=0;
srand(time(NULL));
for (i=0; i<=9; i++)
  {
  for (k=0; k<=9; k++)
    {
      A[i][k]=(99.0*rand()/RAND_MAX)+1.0;
      cout << A[i][k] << " ";
      if (k>i) S=S+A[i][k];
    };
cout << endl;
};
cout << "Сумма элементов выше главной
         диагонали равна" << S;
}  //.
```

5.9. Массивы указателей

Указатели как и переменные любого другого типа, могут объединяться в массивы. Объявление массива указателей из 10 элементов имеет вид `int *x[10];`. Каждому из элементов массива можно присвоить адрес. Например, пятому элементу этого массива присвоим адрес целой и ранее объявленной переменной y: x[4]=&y;. Если затем необходимо найти значение переменной y, это можно сделать, используя конструкцию `*x[4]`.

С помощью массива указателей несложно запрограммировать динамическое создание двумерных массивов. Объявим массив 10 указтелей p:

```
int *p[10];
```

С каждым из них можно связать динамически созданный одномерный массив и интерпретировать его как строку матрицы:

```
for (i=0; i<=8; i++)        //Цикл по строкам
  {p[i]=new int[20];}   // Создание строки.
```

В результате получили следующую замысловатую конструкцию. Идентификатор `p` является именем массива указателей, которые содержат адреса первых элементов других массивов — строк матрицы. Выражение `p[k]` или равносильное ему `*(p+k)` представляют собой адрес первого элемента строки с номером `k`. Выражение `*(p[k]+m)` или `*(*(p+k)+m)` — элемент с номером `m` строки с номером `k`, то есть элемент двумерного массива, расположенный в строке `k` и столбце `m`. Заметим также, что динамически созданные «строки матрицы» не обязательно должны иметь одинаковый размер, а само понятие «матрица» в данном контексте уместно заменить выражением «массив массивов».

5.10. Практическое задание

При выполнении практического задания придерживайтесь следующей последовательности действий.

1. **Обязательно** изучите описание, предлагаемое выше.
2. Разберите примеры, приводимые в описании.
3. Осмыслите задачу, предлагаемую в вашем варианте.
4. Составьте словесный алгоритм решения задачи.
5. Представьте алгоритм решения в виде блок-схемы.
6. Переведите алгоритм на язык программирования.

5.10.1. Варианты заданий

Вариант 1.

1. В одномерном массиве переставьте местами элементы, находящиеся на чётных и нечётных местах. Начальный и преобразованный массивы выведите на экран.

2. В двумерном массиве подсчитайте и выведите на экран максимальные и минимальные элементы каждой строки. Вычислите значение логарифма элемента [1,1] и прибавляйте к нему последовательно все оставшиеся элементы 1-й строки, затем все элементы 2-й строки, начиная с 1-го элемента, и т.д. до тех пор, пока начальное значение (логарифма) не увеличится в 5 раз. Выведите на экран индекс элемента, на котором процесс закончится.

Вариант 2.

1. Преобразуйте одномерный массив таким образом, чтобы сумма элементов в его первой половине была бы как можно ближе к сумме элементов его второй половины. Исходный и преобразованный массивы выведите на экран.

2. В заполненном случайно двумерном массиве a[n][m] замените его элементы вещественными числами, вычисляемыми по формуле a[i][j] = a[i][j] * arccos(i / (i + j)). Для каждого столбца найдите количество положительных и отрицательных элементов в нем. Элементы начального и полученного массивов вывести на экран.

Вариант 3.

1. Элементы случайно заполненного одномерного массива d[n] преобразуйте по формуле d[i] = exp(i/d[i]). В новом массиве найдите и выведите на экран сумму элементов, порядковый номер которых делится на 3.

2. Элементы двумерного массива a[m][n] вычисляются

по формуле $a[i][j] = i^j$, если $i \geq j$ и $a[i][j] = j^i$ — в ином случае. В этом массиве найдите и выведите на экран квадратную матрицу m[3][3], сумма диагональных элементов в которой минимальна.

Вариант 4.

1. Сопоставьте одномерному массиву a[n] другой одномерный массив f[n], элементы которого вычисляются по формуле $f[i] = \ln(a[i])/arctg(e^{i/10})$. Элементы этого массива переставьте таким образом, чтобы меньшему номеру соответствовал бы меньший элемент. Элементы всех массивов выведите на экран в одну строку.

2. В двумерном массиве найдите и выведите на экран 5 наибольших элементов с указанием их индексов.

Вариант 5.

1. В одномерном массиве переставьте элементы таким образом, чтобы на 5-ом месте стояло бы наибольшее, а на 6-ом — наименьшее число.

2. В двумерном массиве b[n][m] найдите 4 последовательные элемента, сумма которых максимальна. Обход для поиска нужно выполнять по ходу часовой стрелки, начиная с элемента b[0,0].

Вариант 6.

1. Выясните, имеется ли среди элементов двумерного массива простые числа. Сообщите их индексы, подсчитайте общее количество и среднее значение.

2. В одномерном массиве найдите среднее значение всех элементов и постройте новый одномерный массив из разностей соответствующих элементов и полученного среднего. Результаты выведите на экран.

Вариант 7.

1. Замените в одномерном массиве каждый из элементов суммой его делителей.

2. В двумерном массиве найдите квадрат размером 3x3, в котором сумма диагональных элементов максимальна. Выведите его на экран.

Вариант 8.

1. Каждый из элементов одномерного массива замените суммой цифр этого элемента. Найдите максимальный и минимальный элементы. Исходный и преобразованный массивы выведите на экран.

2. Преобразуйте исходный двумерный массив a[m][n] в одномерный f[m*n] и, считая каждый из элементов длиной стороны треугольника, найдите тройку таких элементов, которые образуют треугольник с наибольшей площадью.

Вариант 9.

1. В двумерном массиве замените каждый элемент частным от деления самого́ элемента на сумму его делителей.

2. Найдите все цифры в элементах одномерного массива и подсчитайте, сколько раз встречается каждая.

Вариант 10.

1. Элементы одномерного массива преобразуйте в новый одномерный массив, в котором каждый элемент равен наибольшему делителю исходного, а элементы простые числа оставлены без изменений. Оба массива выведите на экран.

2. Для каждой строки двумерного массива подсчитайте и выведите на экран количество элементов, кратных 3 и кратных 7.

Вариант 11.

1. В двумерном массиве переставьте строки таким образом, чтобы среднее значение элементов каждой строки возрастало по мере возрастания её номера.

2. Выясните, содержатся ли в одномерном массиве совершенные числа. Преобразуйте этот массив таким образом, чтобы все элементы, не являющиеся совершенными числами, были заменены суммами их делителей. Исходный и преобразованный массивы выведите на экран.

Вариант 12.

1. Создайте одномерный целочисленный массив. Упорядочите массив по возрастанию элементов методом пузырька.

2. Найдите произведение двух целочисленных матриц 3x3. В полученной матрице перестановкой строк и столбцов добейтесь, чтобы минимальный элемент имел координаты [1][1].

Вариант 13.

1. Сформируйте вещественную матрицу 5x5. Вычислите произведение отрицательных элементов, находящихся над и под главной диагональю матрицы.

2. Упорядочите одномерный вещественный массив из 20 элементов по убыванию методом перебора.

Вариант 14.

1. Сформируйте вещественную матрицу 5x5. Упорядочите по возрастанию элементы каждой строки матрицы.

2. Вычислите сумму квадратов нечётных элементов целочисленного массива из 20 элементов.

Вариант 15.

1. Введите целочисленный двумерный массив из N строк и N столбцов. Найдите максимум среди сумм элементов диагоналей, параллельных побочной диагонали.

2. Задан одномерный массив из N вещественных чисел. Определите образуют ли элементы массива возрастающую последовательность. Если последовательность не возрастает, то упорядочите её по возрастанию любым способом.

Вариант 16.

1. Задан одномерный массив. Определите сколько раз меняется знак в данном массиве, определите номера́ позиций, в которых меняется знак.

2. Упорядочите матрицу построчно по возрастанию.

Вариант 17.

1. Заданы два одномерных массива с N и M элементами, упорядоченными по неубыванию, объедините элементы этих массивов в один массив так, чтобы элементы снова оказались упорядоченными по неубыванию.

2. Осуществите поворот матрицы против часовой стрелки на 90 градусов.

Вариант 18.

1. Дан одномерный массив A, состоящий из N элементов. Выясните, сколько значений элементов в массиве A встречается более одного раза?

2. Определите, является ли заданная матрица ортонормированной, т.е. скалярное произведение каждой пары различных строк (или столбцов) равно нулю.

Вариант 19.

1. Дан одномерный массив A, состоящий из N элементов. Подсчитайте максимальное количество подряд идущих нулей.

2. Заполните квадратную матрицу B[10][10] единицами и нулями в шахматном порядке.

Вариант 20.

1. Дан одномерный массив A, состоящий из N элементов. Исключите из массива первый положительный элемент, следующий за максимальным.

2. Для матрицы C[m][n], найдите сумму элементов, больших 3,2. Переставьте столбцы по возрастанию количества отрицательных элементов в столбцах.

Вариант 21.

1. Дан одномерный массив A, состоящий из N элементов. Подсчитайте максимальное количество положительных элементов, заключённых между нулями.

2. Сформируйте матрицу с треугольником Паскаля (биноминальные коэффициенты)

```
1  0   0   0   0  0  0  0
1  1   0   0   0  0  0  0
1  2   1   0   0  0  0  0
1  3   3   1   0  0  0  0
1  4   6   4   1  0  0  0
1  5  10  10   5  1  0  0
1  6  15  20  15  6  1  0
```

Вариант 22.

1. Дан целочисленный массив M[15]. Определите число соседств из двух чисел разного знака.

2. Для массива S[m][n] определите строку с наибольшим количеством отрицательных элементов. Переставьте столбцы по возрастанию суммы положительных элементов столбцов.

Вариант 23.

1. Определите в данном массиве P[m] количество пар соседних чисел, являющихся противоположными.

2. Дана матрица A[m][n]. Определите среднее арифметическое положительных элементов матрицы. Переставьте столбцы по возрастанию значения последнего элемента столбцов.

Вариант 24.

1. Задан массив A[10]. Определить, сколько содержится в нем различных чисел.

2. Дана матрица A[m][m]. Найдите минимальный из элементов, расположенных под главной диагональю. Переставьте столбцы по возрастанию значения второго элемента столбца.

Вариант 25.

1. Дан массив B[n], содержащий большое количество нулевых элементов. Замените все группы подряд встречающихся нулей на один нуль.

2. Дана матрица A[m][m]. Найдите максимальный из элементов, расположенных над главной диагональю. Переставьте строки по возрастанию значения второго элемента строки.

Вариант 26.

1. Дан массив X[K], содержащий большое количество нулевых элементов. Замените группы элементов, состоящие из нечётного количества нулей, на один нулевой элемент, а из чётного — на два.

2. Найдите все цифры в элементах двумерного массива и подсчитайте, сколько раз встречается каждая.

Вариант 27.

1. Дан массив Y[n], содержащий большое количество нулевых элементов. Замените все группы подряд встречающихся нулей на элемент, состоящий из двух цифр, где первая цифра 0, а вторая — количество нулей в группе.

2. Упорядочите двумерный массив по столбцам по возрастанию.

6. ЭЛЕМЕНТЫ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

6.1. Структуры

Структуры являются развитием понятия массив. Массивы представляют собой объединения *однотипных* переменных. Однако зачастую логика программы требует объединения разнотипных переменных. Пусть, например, программа оперирует описаниями радиоэлектронных компонентов, для определённости — интегральных схем. Каждая их них имеет характеристики различных типов: маркировка, количество контактов, тип корпуса, напряжение питания, потребляемая мощность и т.д. Можно объединить эти разнотиптые характеристики в единую переменную — структуру.

Структура — это объединение *разнотипных* переменных. Общий вид описания структуры:

```
struct ИмяСтруктуры {
Тип1 Имя1;
Тип2 Имя2;
...
};  //.
```

Переменные, которые объединены структурой, называются её полями. Это могут быть обычные переменные, массивы, указатели, другие структуры, объединения. Опишем структуру для хранения данных о студенте

```
struct student {
char fio[30];
int kurs;
char group[7];
```

```
};  //.
```

Объявление структуры является оператором, и поэтому после такого объявления должна стоять точка с запятой. Особо отметим, что объявление структуры не приводит к выделению памяти под переменную. Это лишь описание каркаса будущей переменной, её шаблон. С точки зрения компилятора объявление структуры является описанием нового пользовательского типа данных. Для того чтобы объявить переменные типа student, можно записать

```
student stud1, stud2; //.
```

Здесь объявлены две переменные stud1 и stud2. Под каждую из переменных типа структуры выделяется непрерывный участок памяти.

Переменные структурного типа можно создавать динамически. Для этого нужно описать указатель на структуру

```
student *p;
```

а затем создать экземпляр структуры с помощью операции `new`:

```
p = new student; //.
```

Доступ к элементам структуры осуществялется с помощью операции «точка», например, чтобы вывести на экран поле `kurs` структуры `stud2`, можно использовать оператор

```
cout << stud2.kurs;  //.
```

Если обращение к структуре происходит через указатель, то операцию «точка» используют после разыменования указателя `(*p).kurs`. Для этой конструкции предусмотрен более

наглядный аналог `p->kurs` — операция «стрелка» — обращение к полю `kurs` структуры, на которую указывает указатель `p`.

Таким образом, структуры представляют собой способ организации данных в программе. Обработка этих данных (если необходимо) происходит с помощью функций. Дальнейшим развитием способов программирования стало объединение данных с обрабатывающими их подпрограммами и появление понятий «класс» и «объект».

6.2. Классы и объекты

Класс — это такой «тип данных», который представляет собой объединение полей (аналогично структуре) и подпрограмм (функций). Подпрограммы, принадлежащие классу называются методами. Общий вид описания класса похож на описание структуры

```
class ИмяКласса {
Тип1 Имя1;
Тип2 Имя2;
...
Функция1
Функция2
...
};   //.
```

Доступностью включенных в класс элементов можно управлять с помощью служебных слов private, public, protected. Словом public обозначаются элементы класса, доступные для других объектов программы, словом private — элементы, доступные только для методов этого класса, protected — элементы, доступные внутри методов класса и всех его наследников. В качестве примера опишем класс для работы с

комплексными числами. Комплексное число состоит из двух компонент — действительной и мнимой частей. Значит класс должен как минимум содержать два поля типа `float`

```
class CComplex {
public :
  float x;
  float y;
};  //.
```

Однако, такое определение ничем не отличается от структуры. Расширим класс двумя методами для вычисления модуля и аргумента комплексного числа

```
class CComplex {
public :
  float x;
  float y;
float Modul(void)
  {return sqrt(x*x+y*y);}

float Argum(void)
  {if (x==0) return 3.14159/2;
     else return atan(y/x);}
};  //.
```

Здесь реализация обоих методов приведена внутри описания класса. Такой способ является удобным только для небольших классов. Если методов достаточно много, целесообразно отделить описание класса от его реализации. Тогда в описании указывается только прототип метода, например,

```
class CComplex {
public :
```

```
  float x;
  float y;
float Modul(void);
float Argum(void);
};
```

и оно размещяется в заголовочном файле `complex.h`. Реализацию методов можно разместить в отдельном файле `complex.cpp` и оформить следующим образом:

```
#include "complex.h"

float CComplex::Modul(void)
  {return sqrt(x*x+y*y);}

float CComplex::Argum(void)
  {if (x==0) return 3.14159/2;
     else return atan(y/x);}  //.
```

Также как при объявлении структуры, определение класса не приводит к выделению памяти. Оно является описанием нового типа, который вместе с данными обладает подпрограммами их обработки. Переменные этого типа называются объектами. Создадим объект с именем A, представляющий собой комплексное число

```
CComplex A;  //.
```

Объект — это переменная типа класс. Такая терминология используется не случайно. Объекто-ориентированная технология программирования позволяет строить модели реаальных объектов, а также выстраивать их иерархии с помощью наследования свойств одних объектов их наследниками. В этой связи полезно вспомнить известную из биологии классификацию животных. Так, например, класс млекопитающих обладает некоторыми свойствами, присущими

абслютно всем его представителям. Вместе с тем среди млекопитающих выделяют отряд хищных. Они вдобавок имеют свои свойства. Среди хищных имеется семейство медвежьих со своими свойствами. Таким образом, «конкретный экземпляр» медведь бурый обладает свойствами всех перечисленных уровней классификации. Другими словами, каждый следующий уровень классификации добавляет свои свойства. Медведь бурый является аналогом объекта, а уровни классификации — это классы со структурой наследования.

В качестве примера построим класс, описывающий точку на экране монитора. Чтобы охарактеризовать точку необходимо указать её координаты — два целых числа `a` и `b`. Точка может перемещаться скачком в новые координаты `x`, `y`. Поэтому она должна обладать методом перемещения `move`, меняющим её координаты. Опишем класс точки:

```
class TPoint {
public :
  int a;
  int b;
void move(int x,y);
};

TPoint::move(int x,y)
   {a=x; b=y;}  //.
```

Далее построим ещё один класс, описывающий видимую точку. Помимо координат и возможности перемещения точка должна иметь характеристику видимости `int visible`. Если `visible=0`, то точка невидима, в противном случае — видима. Для управления видимостью предусмотрим два метода `show` и `hide`.

```
class TVisiblePoint  : public TPoint {
public :
  int visible;
void show(void);
void hide(void);
};

TVisiblePoint::show()
   {visible = 1;}

TVisiblePoint::show()
   {visible = 0;}  //.
```

Первая строка описания класса задаёт наследование классом `TVisiblePoint` всех элементов класса `TPoint`. Слово `public` при описании наследования означает, что производному классу `TVisiblePoint` будут доступны все элементы базового класса `TPoint`.

И, наконец, построим ещё один класс, описывающий цветную видимую точку. Очевидно, этот класс будет наследником класса `TVisiblePoint`, только будет иметь своё поле, характеризующее цвет точки. Для простоты будем считать, что цвет задаётся номером целого типа. Тогда приходим к следующему описанию класса

```
class TColorVisiblePoint  : public TVisiblePoint {
public :
  int color;
void setcolor(int c);
};

TColorVisiblePoint::setcolor(int c)
   {color = c;}  //.
```

Здесь метод `setcolor()` служит для перекрашивания точки в другой цвет. Теперь можно создать объект точку.

```
TColorVisiblePoint  ExamplePoint;
```

и использовать её:

```
ExamplePoint.x=10;          // координата x
ExamplePoint.y=10;          // координата y
ExamplePoint.show();        // показать
ExamplePoint.move(50,50);   // переместить
ExamplePoint.setcolor(5);   // перекрасить.
```

На первый взгляд кажется, что использование объектно-ориентированного программирования приводит к излишнему нагромождению кода. Однако, такое впечатление справедливо при программировании достаточно маленьких учебных примеров. При создании объёмных программ, а особенно при работе коллектива программистов применение объектов может дать существенный выигрыш в производительности труда программиста. Вообще считается, что нецелесообразно абсолютно все программы делать объектными. Объектную технологию нужно применять там, где это удобно.

6.3. Конструкторы и деструкторы

При создании объекта зачастую необходимо задать начальные значения его полям, выполнить начальную инициализацию объекта. Для этого служит специальный метод, называемый конструктором. Его имя *всегда* совпадает с именем класса. Конструктор может зависеть от параметров, которые можно использовать как начальные значения полей. Напишем конструктор для класса `TColorVisiblePoint`

```
TColorVisiblePoint::TColorVisiblePoint(int X,Y,C)
   {x=X; y=Y; color = C;}  //.
```

Здесь заданы начальные значения координат и цвет точки. Теперь при создании объекта необходимо указывать эти начальные значения, например так:

```
// создаём
TColorVisiblePoint  *ExamplePoint =
     new TColorVisiblePoint(10,10,5);
ExamplePoint->show();       // показываем
ExamplePoint->move(50,50);  // перемещаем.
```

Деструктор выполняет функцию, обратную конструктору. При удалении объекта вызывается метод-деструктор, имя которого совпадает с именем класса, но вначале приписана тильда ~. В основном деструкторы применяют в том случае, когда выполнялось динамическое создание внутрениих объектов или динамическое выделение памяти. Тогда внутри деструктора необходимо освободить зарезервированную память.

6.4. Инкапсуляция, наследование, полиморфизм

Объектно-ориентированные языки программирования обеспечивают механизмы, которые помогают реализовывать объектно-ориентированную модель. К ним относятся *инкапсуляция, наследование и полиморфизм.*

Инкапсуляция — это механизм защиты данных от несанкционированного изменения. При описании класса программист выделяет части класса, доступные внешней программе, другим объектам (public) и недоступные им (private

и protected). Целесообразно как можно больше полей делать закрытыми, а для взаимодействия с ними предусматривать открытые методы. При такой организации класса говорят, что закрытые части инкапсулированы в классе. Считается, что инкапсуляция приводит к увеличению надёжности программ, запрещая несанкционированное изменение данных. Кроме того, инкапсуляция позволяет выделить интерфейсную часть класса, предназначенную для взаимодействия объекта с окружающей средой и реализацию класса. При неизменной интерфейсной части реализация класса может меняться от версии к версии, не влияя на работоспособность остальных компонент приложения.

Наследование — это механизм такой организации отношений между классами, когда один класс может использовать поля и методы другого класса, а в результате один объект приобретает свойства другого. Родительский класс называют базовым, дочерний — производным.

Полиморфизм — это механизм использования одинаковых имен для методов, объектов и т.д. Существует несколько проявлений полиморфизма в объектно-ориентированном программировании. Первое из них связано с применением одинаковых имён методов в родительском и дочернем классах. В этом случае говорят, что дочерний метод перекрывать родительский. Какой из методов будет вызван зависит от контекста его использования. Второе проявление также связано с применением одинаковых имён методов, но имеющих разные типы параметров и результатов. Это позволяет найти третье применение полимирфизма — хранение данных разных типов в полях объекта с одинаковыми именами.

Подробнее о механизмах инкапсуляции, наследования и полиморфизма можно почитать в [6] – [9].

7. ЛАБОРАТОРНАЯ РАБОТА №3 РАБОТА С ТЕКСТОМ, ФАЙЛАМИ

Цели работы:

- освоение приёмов работы с текстовыми данными;
- освоение приёмов работы с файлами.

7.1. Работа со строками

7.1.1. Работа со строками средствами языка С

Большинство программ содержат элементы работы с текстовыми данными. Текстовые данные состоят из строк. Строка представляет собой массив символов. Поэтому массивам типа char — символьным массивам — следует уделить особое внимание при изучении программирования. Во многих языках программирования есть специальный тип данных — строка символов (string). В языке С такого типа нет, а работа со строками реализована путём использования одномерных массивов типа char. В языке С символьная строка — это одномерный массив типа char, заканчивающийся нулевым байтом. Для нулевого байта определена специальная символьная константа `\0`. Это необходимо учитывать при описании соответствующего символьного массива. Так, если строка должна содержать N символов, то в описании массива необходимо указать N+1 элемент. Например, описание `int str[12]`, предполагает, что строка содержит 11 символов, а последний байт зарезервирован под нулевой байт.

Хотя в языке С нет специального типа строки, язык допускает строковые константы. Строковая константа — это последовательность литер, заключенных в двойные кавычки. Например, "Это строковая константа". В конец строковой константы не надо ставить символ `\0`. Это сделает сам

компилятор, и строка «CodeBlocks» в памяти будет выглядеть так:

...	C	o	d	e	B	l	o	c	k	s	\0	...

Есть два простых способа ввести строку с клавиатуры. Первый способ — воспользоваться функцией `scanf()` со спецификатором ввода `%s`. Надо помнить, что функция `scanf()` вводит символы до первого пробельного символа. Второй способ — воспользоваться функцией `gets()`, объявленной в файле `stdio.h`. Функция `gets()` позволяет вводить строки, содержащие пробелы. Ввод заканчивается нажатием клавиши Enter. Обе функции автоматически добавляют в конце строки нулевой байт. В качестве параметра в этих функциях используется имя массива.

Вывод строк производится с помощью функций `printf()` и `puts()`. Обе функции выводят содержимое массива до первого нулевого байта. Функция `puts()` добавляет в конце выводимой строки символ новой строки. В функции `printf()` переход на новую строку нужно предусматривать в строке формата самим. Рассмотрим пример:

```
# include <stdio.h>
// Ввод строки с клавиатуры и вывод ее на экран
int main(void)
{
char str[80];
printf("Введите строку длиной не более 80 символов");
gets(str);
printf(" Вы ввели строку %s\n",str);
printf("Введите еще одну строку ");
scanf("%s",str);
printf(" Вы ввели строку ");
puts(str);
} //.
```

Для работы со строками существует специальная библиотека, заголовочный файл которой называется `string.h`. Перечислим некоторые функции этой библиотеки.

Функция `strcpy(sl, s2)` используется для копирования содержимого строки `s2` в строку `s1`. Строка `s1` должна быть достаточно большой, чтобы в неё поместилась строка `s2`. Если места мало, то компилятор не выдаст соответствующего сообщения об ошибке, также это не приведёт к прерыванию выполняемой программы, но может привести к порче других данных, что отразится на результате.

Функция `strcat(sl, s2)` присоединяет строку `s2` к строке `s1` и помещает её в строку, `s1`, при этом строка `s2` не изменяется. Нулевой байт, который завершал строку `s1`, будет заменён первым символом строки `s2`. И в функции `strcpy()`, и в функции `strcat()` полученная строка автоматически завершается нулевым байтом. Рассмотрим пример использования этих функций:

```
# include <stdio.h>
# include <string.h>
int main(void)
{
char s1[20], s2[20];     // описали массивы
strcpy(sl," Hello,");    // s1="Hello,"
strcpy(s2," World !");   // s2="World !"
puts(sl); puts(s2);      // выводим Hello, World !
strcat(sl,s2);           // s1="Hello, World !"
puts(sl);                // выводим Hello, World !
}                        //.
```

Вызов функции `strcmp()` имеет вид `strcmp(sl, s2)`. Эта функция сравнивает строки `s1` и `s2` и возвращает нулевое значение, если строки равны, т. е. содержат одно и то

же число одинаковых символов. Под сравнением строк понимается сравнение в лексикографическом смысле. Если `s1` лексикографически больше `s2`, то функция `strcmp()` возвращает положительное значение, если меньше — отрицательное.

Функция `strlen(s)` возвращает длину строки `s`, при этом завершающий нулевой байт не учитывается. Например, вызов `strlen("hello")` вернёт значение 5.

7.1.2. Массивы строк

Поскольку строка представляет собой массив символов, то массив строк является двумерным массивом символов со строками разной длины. Для работы с подобного рода данными удобно использовать массивы указателей. Каждый из указатетей массива может содержать адрес своего массива символов, то есть своей строки. Рассмотрим пример использования массива строк, заимствованный из [4]:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <conio.h>
int main(void)
{
// описываем массив строк
char *ext[]={"exe","com","dat","c","pas","cpp"};
char ch, s1[80];
for( ; ; ) // бесконечный цикл
{do
{
// Выводим подсказки на экран
printf(" Файлы с расширением:\n");
printf("1. exe\n");
```

```
printf("2. com\n");
printf("3. dat\n");
printf("4. c\n");
printf("5. pas\n");
printf("6. cpp\n");
printf("7. Exit\n");
printf(" Ваш выбор: \n");
// Ввод выбранного символа 1-7
ch=getche();
printf("\n"); }
// пока не будет нажат символ 1-7
while ((ch<'1')||(ch>'7'));
// анализ введённого символа
if ( ch == '7' ) break;
strcpy(s1,"dir *.");
strcat(s1,ext[ch-49]);
system(s1);
}
} //.
```

Данная программа формирует командную строку для выполнения команды «dir» (вывод на экран списка файлов текущего каталóга) и затем выполняет её с помощью библиотечной функции `system()`. Пользователь может выбрать расширение выводимых файлов из тех, которые предоставляет программа. Возможные расширения хранятся в виде массива строк и выводятся на экран как подсказка. Выбор нужного расширения осуществляется вводом его номера в списке. Далее программа анализирует код введённого символа. Символы цифр от 1 до 7 имеют коды 49 ... 55. Следовательно, ввод единицы соответствует символу с кодом 49 и должен привести к выбору из массива расширения «exe» — элемент массива с номером 0. Если пользователь хочет

вывести файлы с расширением `cpp` , он указывает символ 6, код которого равен 54. Разность 54 – 49 определяет индекс шестого элемента в массиве `ext`. Подобным образом используются массивы указателей для хранения сообщений о возможных ошибках [4]:

```
char *errors[]= {"Cannot open file",
"Cannot close file", "Allocation error",
"System error" };  //.
```

При таком объявлении строчные константы располагаются в памяти в разделе констант, массив указателей состоит из четырёх элементов, для которых тоже выделяется память. Значениями этих элементов будут являться адреса, указывающие на начало строковых констант.

7.1.3. Класс string библиотеки C++

Стандартная библиотека языка C++ содержит описание класса `string`, который предоставляет богатый набор возможностей для работы со строками. Объект класса `string` инкапсулирует строку в виде массива символов, а также обладает исчерпывающим набором методов для работы со строкой. Для использования класса необходимо включить в текст программы заголовочный файл

```
#include <string>
```

после чего можно создавать объект класса `string`

```
string stroka( "Это строка\n" );
```

или с помощью указателя

```
string *stroka = new string( "Это строка\n" );  //.
```

Здесь конструктору класса передаётся строковая константа для начальной инициализации строки. Если начальная инициализация отсутствует, то создаётся пустая строка. Строке можно присваивать строковое значение

```
stroka = "Другая строка\n";  //.
```

Чтобы вывести строку на экран можно отправить её в поток `cout`

```
cout << stroka;  //.
```

Однако вывести её с помощью функции `printf` можно только после преобразования в стандартный массив символов, оканчивающийся нулём

```
printf("Выводим строку %s", stroka.c_str());  //.
```

Здесь метод `c_str()` возвращает нужный массив символов. Доступ к отдельным элементам строки — символам — возможен через операцию взятия индекса, например,

```
stroka[5]='!';  //.
```

По сути все действия со строкой реализованы в виде методов класса `string`. Поэтому перечислим имена и назначение основных методов этого класса:

- `begin()` — возвращает указатель на первый символ строки,
- `end()` — возвращает указатель на воображаемый символ, следующий за последним,
- `append(строка)` — добавляет символ или строку в конец строки,
- `at(номер)` — возвращает символ, находящийся в указанной позиции,

- `c_str()` — возвращает строку в виде массива символов с завершающим нулём,
- `clear()` — удаляет все символы строки, оставляя её пустой,
- `compare(строка)` — сравнивает две строки,
- `data()` — возвращает указатель на первый символ строки,
- `empty()` — проверяет строку на пустоту,
- `erase()` — удаляет символы из указанных позиций,
- `find(подстрока)` — возвращает номер позиции первого появления подстроки в строке,
- `find_first_not_of(строка)` — находит номер позиции первого символа в строке, не входящего в набор символов другой строки,
- `find_first_of(строка)` — находит номер позиции первого символа в строке, входящего в набор символов другой строки,
- `find_last_not_of(строка)` — находит номер позиции последнего символа в строке, не входящего в набор символов другой строки,
- `find_last_of(строка)` — находит номер позиции последнего символа в строке, входящего в набор символов другой строки,
- `getline` — считывает строку из потока ввода,
- `insert(номер, подстрока)` — вставляет подстроку в указанную позицию строки,
- `length` — возвращает длину строки
- `max_size` — возвращает максимально возможный размер строки,
- `rfind` — осуществляет поиск последнего появления подстроки в строке,
- `size` — возвращает длину строки.

Отдельного внимания заслуживает функция `replace` (не путайте с методами). Она позволяет заменить один или несколько символов строки другими символами. Как правило она используется в следующем контексте

```
replace(stroka.begin(),stroka.end(),'5','*');  //.
```

Здесь символ '5' меняется на звёздочку во всей строке.

7.2. Работа с файлами

7.2.1. Открытие файлов

Программирование взаимодействия с файлами необходимо почти во всех практически значимых программах. Основные функции для работы с файлами описаны в библиотеке `stdio` и представляют собой операции открытия файла, чтения или записи в файл, закрытия файла. Для доступа к файлу применяется тип данных FILE, указатель на который будет использоваться для связи с файлом и служить так называемой «файловой переменной». Таким образом, вначале необходимо описать указатель f на переменную типа FILE.

```
FILE *f;  //.
```

Далее нужно открыть файл. Для этого используется функция `fopen()`. В результате своей работы функция `fopen()` динамически создаёт в памяти переменную типа FILE и возвращает её адрес. Аргументами для `fopen()` служат две строки. Первая — путь к файлу, построенный по правилам операционной системы. Вторая строка содержит символы, отражающие режим открытия файла. Символ 'r' (read) означает, что файл открывается для чтения. Символ 'w' (write) означает открытие файла для перезаписи. Старое содержимое файла бесследно теряется, а в случае отсутствия файла

он создаётся. Символ 'a' (append) означает открытие файла для добавления. Если файл не существует, он создаётся. Символы 't' и 'b' используются для открытия текстовых и бинарных файлов соответственно. В следующем примере выполняется открытие трёх файлов.

```
FILE *f1, *f2, *f3;
f1 = fopen("temp.txt", "rt");
f2 = fopen("z:/Prog/file.cpp", "w");
f3 = fopen("z:\\Prog\\file.cpp", "at");
```

Текстовый файл `temp.txt` открывается для чтения и предполагается, что он находится в одной папке с программой. Файл `file.cpp` открывается двумя способами. В первом из них для разделения папок в пути используется прямой слеш /, а во втором — удвоенный обратный слеш (согласно правилам языка C) .

При неудачной попытке открытия файла, например, при открытии на чтение несуществующего файла, функция `fopen()` возвращает нулевой указатель NULL. Поэтому после команды открытия файла целесообразно проверить полученный указатель:

```
#include <stdio.h>
...
FILE *f = fopen("file.txt", "rt");
if (f == NULL)
{ printf("Ошибка открытия файла!!!"); }
...
```

Помимо собственного сообщения об ошибке можно воспользоваться также системным сообщением, в котором более точно указаны причины ошибки. Для этого вместо `printf()` необходимо воспользоваться функцией `perror()`

```
perror("Ошибка открытия файла!!!"); //.
```

Сначала на экран выведется сообщение «Ошибка открытия файла!!!», а затем — системное сообщение об ошибке.

7.2.2. Текстовые и бинарные файлы

С точки зрения программиста файл представляет собой потенциально бесконечную ленту записей какого-либо размера. Однако видны не все записи одновременно, а только их часть, как будто смотрим на ленту через воображаемое окно, называемое указателем (не путать с указателем FILE *). В языке С принято считать, что размер окна равен одному байту. При считывании или записи указатель преремещается по ленте. В конце файла находится так называемый «признак конца файла», обозначаемый в языке С константой EOF.

Числовые данные, которыми оперирует программа хранятся в памяти ЭВМ с том формате, который предусмотрен типом этих данных. Например, для переменной целого типа `int` выделяются 4 байта. Данные хранятся в дополнительном коде. При выводе значения переменной на экран она переводится в текстовый, понятный человеку вид, то есть в последовательность символов. Точно также в файле значение переменной может храниться либо в двоичном виде (как в памяти ЭВМ), либо в текстовом — как последовательность символов. В связи с этим файлы делятся на двоичные (бинарные) и текстовые. При работе с бинарными файлами операции чтения и записи не требуют никакого преобразования данных, а при работе с текстовыми файлами требуется преобразовывать данные из текстового или в текстовый вид.

7.2.3. Чтение из файла

Чтение бинарных файлов.

Чтение бинарных данных из файла производит функция `fread()`, которая имеет 4 аргумента: имя массива, в который будет производиться чтение, размер одного элемента массива, число элементов для чтения, указатель файловой переменной. Возвращаемое функцией `fread()` значение равно реальному количеству считанных элементов. Пусть, например, необходимо считать из файла 512 вещественных чисел типа `float`. Предусмотрим массив `mas`, в который помещаются данные, переменную `kolvo` для результата считывания и указатель на файловую переменную

```
unsigned int kolvo;
float mas[512];
FILE *f;  //.
```

После открытия файла выпоняем считывание

```
kolvo = fread(mas, sizeof(float), 512, f);
```

Здесь функция `sizeof(float)` вычисляет размер считываемого элемента, а переменная `kolvo` принимает значение, равное реальному количеству считанных элементов. Её значение может быть меньше предусмотренных 512 элементов, поскольку неожиданно может встретиться признак конца файла.

Чтение текстовых файлов.

Чтение текстовых файлов можно осуществить с помощью функции форматного ввода `fscanf()`. Она преобразует считанные данные из текстового представления в бинарное. Использование функции `fscanf()` аналогично функции ввода с клавиатуры `scanf()`, только первым аргументом должен быть указатель на файловую переменную

```
scanf (Указатель на файл,
"Управляющая строка",Арг1,Арг2,...); //.
```

Вот несколько примеров использования функции `fscanf()`:

```
int K, kolvo;
float X, Y;
char C;
char S[80];
FILE *f;
    . . .
    fscanf(f, "%d", &K);
    fscanf(f, "%f", &X);
    fscanf(f, "%c", &C);
    fscanf(f, "%s", S);
    fscanf(f, "%f%f", &X, &Y);
    . . .
```

Переменная `kolvo` принимает значение, равное количеству введённых данных.

7.2.4. Запись в файл

Запись бинарных данных.

Запись бинарных данных в файл можно осуществить с помощью функции `fwrite()`. Она имеет 4 аргумента: имя массива, значения элементов которого необходимо записать в файл, размер одного элемента массива, число записываемых элементов, указатель файловой переменной. Возвращаемое функцией `fwrite()` значение равно реальному количеству записанных элементов. Пусть, например, необходимо записать в файл 512 вещественных чисел типа `float` из массива `mas`

```
unsigned int kolvo;
```

```
float mas[512];
FILE *f;  //.
```

После открытия файла выполняем запись

```
kolvo = fwrite(mas, sizeof(float), 512, f);  //.
```

Здесь функция `sizeof(float)` вычисляет размер записываемого элемента, а переменная `kolvo` принимает значение, равное реальному количеству записанных элементов. Её значение может быть меньше предусмотренных 512 элементов, поскольку неожиданно может закончиться место на диске.

Запись в текстовый файл.

Запись бинарных данных в текстовый файл можно осуществить с помощью функции форматного вывода `fprintf()`. Она преобразует записываемые данные в текстовое представление в соответствие с форматной строкой. Использование функции `fprintf()` аналогично функции вывода на экран `printf()`, только первым аргументом должен являться указатель на файловую переменную.

```
printf (Указатель на файл,
"Управляющая строка",Арг1,Арг2,...);  //.
```

Управляющая строка может содержать компоненты трёх типов:
1) обычные символы, которые просто выводятся на экран;
2) управляющие константы из табл. 5;
3) спецификации преобразования.
Каждая спецификация преобразования служит для вывода на экран очередного данного из списка аргументов Арг1, Арг2, ... Начинается спецификация знаком %, заканчивается — символом преобразования. В следующем примере

```
fprintf(f,"Результат вычислений равен %12.3f\n", y);
```

символом преобразования служит буква `f`. Следовательно, функция `printf()` выведет на экран вместо %12.3f значение переменной `y`, которая стоит в списке аргументов. При этом всего для значения переменной `y` будет предусмотрено 12 знаков, среди них 3 — для дробной части.

7.2.5. Закрытие файла

По окончании работы с файлом его необходимо закрыть. Для закрытия файла используется функция `fclose()`, аргументом которой является указатель на файловую переменную, например

```
fclose(f);  //.
```

При удачном закрытии функция возвращает ноль, а при ошибке — значение -1. Поэтому при закрытии файла можно использовать следующего вида проверку

```
if (fclose(f) < 0)
{
  printf("Невозможно закрыть файл!!!");
} //.
```

7.2.6. Позиционирование указателя файла

Нередко встречается необходимость считывать и записывать данные в файл непоследовательно. При чтении или записи нескольких байт указатель текущей позиции автоматически смещается на нужное число позиций. Существует также функция принудительного позиционирования указателя текущей позиции в нужное место `fseek()`. Эта функция имеет три аргумента: указатель на файловую переменную

FILE *f, величина смещения целого знакового типа, величина целого типа, характеризующая точку отсчёта смещения. Для третьего аргумента предусмотрены три возможных значения, заданные как целые константы: SEEK_CUR=1 — смещение отсчитывается от текущей позиции указателя, SEEK_SET=0 — смещение отсчитывается от начала файла, SEEK_END=2 — смещение отсчитывается от конца файла. Рассмотрим несколько примеров:

```
fseek(f, 0, SEEK_SET);
fseek(f, -8, SEEK_END);
fseek(f, 5, SEEK_CUR);  //.
```

Первая из команд позиционирует указатель в начало файла, вторая — в восьми байтах от конца файла, третья команда смещает указатель на 5 байт вперёд.

Функция `ftell()` позволяет узнать номер текущей позиции указателя, например,

```
N=ftell(f);  //.
```

Функция `feof()` даёт возможность проверить, достиг ли указатель конца файла. Она возвращает ненулевое значение, если конец файла достигнут.

7.2.7. Понятие потока ввода-вывода

Все современные операционные системы поддерживают так называемый механизм потоков. Поток представляет собой последовательность байт. Он всегда имеет только одно направление и по отношению к программе является потоком или ввода, или вывода. Программа может читать данные из потока ввода и выводить данные в поток вывода. По сути работа с потоком ничем не отличается от работы с файлом. Поэтому для реализации потоков в языке С используются

принципы взаимодействия с файлами. Доступ к потоку осуществляется с помощью переменной типа FILE * (указатель на файловую переменную). Существуют три стандартных потока, подключаемые к каждой прогрмме: `*stdin` — входной поток, связанный с клавиатурой; `*stdout` — выходной поток, связанный с экраном; `*stderr` — выходной поток для вывода сообщений об ошибках, тоже связанный с экраном. Таким образом, следующие команды вывода сообщения на экран равносильны

```
printf("Сообщение");
fprintf(stdout,"Сообщение");  //.
```

7.2.8. Функции библиотеки ввода-вывода

Помимо перечисленных выше библиотека ввода-вывода stdio содержит ещё много полезных функций. Далее в качестве справочной информации приведены прототипы некоторых из них.

Функция `remove` удаляет файл с именем `filename`

```
int remove(char *filename);  //.
```

Функция `rename` переименовывает файл `old` в файл `new`

```
int rename(char *old, char *new);
```

В табл. 8 приведены функции ввода-вывода бинарных данных с кратким описанием и прототипом.

Таблица 8

Функции ввода-вывода бинарных данных

Для любого потока	
fgetc	Считывает один символ из потока A, int fgetc(FILE *A)
fgets	Считывает строку s длиной не более n символов из потока A, char *fgets(char *s, int n, FILE *A)
fputc	Выводит символ C в поток A, int fputc(int C, FILE *A)
fputs	Выводит строку символов S в поток A, int fputs(char *S, FILE *A)
getc	Считывает из потока A один символ, int getc(FILE *A)
putc	Выводит символ C в поток A, int putc(int C, FILE *A)
putw	Выводит в поток A целое значение W, int putw(int W, FILE *A)
getw	Считывает из потока A целое число, int getw(FILE *A)
Для потока stdin	
gets	Считывает строку символов S из потока stdin, char *gets(char *S)
getchar	Считывает символ из потока stdin, int getchar(void)
Для потока stdout	
putchar	Выводит символ C в поток stdout, int putchar(int C)
puts	выводит строку S в поток stdout, int puts(const char *S)

7.2.9. Классы и объекты потоков ввода-вывода

В C++ применяется объектно-ориентированный подход к реализации обмена данными с потоками ввода-вывода. Стандартная библиотека языка C++ iostream содержит описания классов, инкапсулирущих потоки ввода-вывода. Базовым для классов потоков является класс `ios`. Его наследники — классы `istream` и `ostream` — отвечают за ввод и вывод соответственно. Они имеют общего наследника — класс `iostream`. При запуске программы автоматически создаются объекты `cin` класса `istream` и объекты `cout`, `cerr`, `clog` класса `ostream`. Первый из них связан с клавиатурой и поддерживает операцию взятия из потока `>>`. Остальные связа-

ны с экраном и поддерживают операцию помещения в поток <<.

Для организации файлового ввода-вывода классы `istream`, `ostream` и `iostream` обладают наследниками `ifstream`, `ofstream` и `fstream` соответственно. Перечисленные классы имеют множество методов для управления вводом и выводом и могут использовать так называемые манипуляторы. К примеру, целые числа обычно интерпретируются как десятичные. Поместив в поток `cout` манипулятор `hex`, получим на экране числа в шестнадцатеричной системе

```
cout << "N=" << hex << N << endl; //.
```

Аналогично работают манипуляторы `oct` и `dec`. Точность вывода в поток вещенственных чисел регулирует манипулятор `precision(m)`, где `m` — количество знаков после запятой. При `m=0` устанавливается точность по умолчанию (шесть знаков). Подробнее о форматировании ввода-вывода можно почитать в [1] стр. 630 – 672.

Использование файлового ввода-вывода становится доступным после включения в программу заголовочного файла `fstream`. Чтобы открыть файл на чтение необходимо создать объект класса `ifstream` и передать имя открываемого файла его конструктору в качестве параметра

```
ifstream fileIn("d:\\text\\fileIn.txt");  //.
```

Аналогично выполняется открытие файла для записи, только используется объект класса `ofstream`

```
ofstream fileOut("d:\\text\\fileOut.txt");  //.
```

Имена объектов `fileIn` и `fileOut` можно использовать наряду со стандартными потоками `cin` и `cout`, например,

```
for(k=1;k<=100;k++) {fileOut << x[k];}  //.
```

Уточнить, как именно следует отрывать файл, можно с помощью второго (необязательного) аргумента конструктора. Для этого можно использовать следующие константные значения `ios::app` — открытие файла на добавление, `ios::ate` — на добавление с возможностью перемещаться по файлу, `ios::trunc` — на перезапись (прошлое содержимое теряется), `ios::nocreate` — если файл не существует, то открытие не выполняется, `ios::noreplace` — если файл существует, то открытие не выполняется. Так, например, для открытия файла на добавление необходимо писать

```
ifstream fileOut("d:\\text\\fileOut.txt", ios::app);
```

После использования файл необходимо закрыть с помощью метода `close()`.

Подробнее об использовнии потоковых объектов можно прочитать в [2] стр. 505–542.

7.3. Практическое задание

При выполнении практического задания придерживайтесь следующей последовательности действий.

1. **Обязательно** изучите описание, предлагаемое выше.
2. Разберите примеры, приводимые в описании.
3. Осмыслите задачу, предлагаемую в вашем варианте.
4. Составьте словесный алгоритм решения задачи.
5. Представьте алгоритм решения в виде блок-схемы.
6. Переведите алгоритм на язык программирования.

7.3.1. Варианты заданий

Вариант 1.

Напишите функцию приближённого вычисления квадратного корня по формуле $x_n = x_{n-1}/2 + a/2x_{n-1}$, $x_1 = 1$, $x \to \sqrt{a}$. Найдите корни всех чисел из входного текстового файла, добавить их в тот же файл рядом с исходными.

Вариант 2.

Дано предложение, слова в нём разделены одним или несколькими пробелами. Предложение записано в текстовом файле. Замените каждое слово количеством букв ‘а’ в нём. Используйте функцию подсчёта количества заданной буквы в слове.

Варианты 3-18.

В текстовом файле находятся записи о номерах телефонов. В каждой строке записана информация об одном абоненте: № телефона, ФИО, адрес. Задачи необходимо решить с использованием процедур и функций.

3. Напишите программу, которая осуществляет добавление записи в файл с проверкой корректности данных.

4. Напишите программу, которая удаляет из файла указанную запись.

5. Напишите программу, которая согласно запросу пользователя выводит информацию об абонентах, фамилия которых начинается с указанной буквы.

6. Напишите программу, которая согласно запросу пользователя выводит информацию об абонентах, подключенных к одной АТС.

7. Напишите программу, которая осуществляет поиск абонента с заданной фамилией.

8. Напишите программу, которая осуществляет поиск

абонента с заданным именем.

9. Напишите программу, которая осуществляет поиск абонента с заданным номером.

10. Напишите программу, которая для каждой АТС выводит процент заполненности телефонных номеров.

11. Напишите программу, которая выводит диапазоны свободных номеров телефонов для каждой АТС.

12. Напишите программу, которая сортиру́ет записи в файле в порядке возрастания номера телефона.

13. Напишите программу, которая сортиру́ет записи в файле в алфавитном порядке фамилий.

14. Напишите программу, которая согласно запросу пользователя выводит информацию об абонентах, проживающих на указанной улице.

15. Напишите программу, которая согласно запросу пользователя выводит информацию об абонентах, проживающих в указанном доме.

16. Напишите программу, которая согласно запросу пользователя выводит информацию об абонентах, с указанными фамилией и именем.

17. Напишите программу, которая согласно запросу пользователя выводит информацию об абонентах, фамилия которых заканчивается указанной буквой.

18. Напишите программу, которая выводит диапазоны занятых номеров телефонов для каждой АТС без указания информации об абонентах.

Варианты 19-27.

В текстовом файле находятся записи о книгах в библиотеке. В каждой строке записана информация об одной книге: ФИО автора, название, год издания, количество страниц. Задачи необходимо решить с использованием процедур

и функций.

19. Напишите программу, которая осуществляет добавление записи в файл с проверкой корректности данных (не существует ли уже добавляемая запись).

20. Напишите программу, которая удаляет из файла указанную запись.

21. Напишите программу, которая согласно запросу пользователя выводит информацию о книгах указанного автора.

22. Напишите программу, которая согласно запросу пользователя ищет книгу по указанной части названия.

23. Напишите программу, которая сортиру́ет записи в файле в алфавитном порядке фамилий авторов.

24. Напишите программу, которая согласно запросу пользователя выводит информацию о книгах, выпущенных после указанного года.

25. Напишите программу, которая согласно запросу пользователя выводит информацию о книгах больше заданного числа страниц.

26. Напишите программу, которая согласно запросу пользователя выводит информацию о книгах, фамилия одного из авторов которых заканчивается указанной буквой.

27. Напишите программу, которая вычисляет, в какой год было выпущено максимальное количество книг.

8. ЛАБОРАТОРНАЯ РАБОТА №4 СОЗДАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ В ТЕКСТОВОМ РЕЖИМЕ

Цели работы:

- ознакомление с принципами построения интерактивных программ;
- ознакомление с библиотекой PDCurses;
- освоение принципов программирования меню;
- освоение приёмов работы с окнами в текстовом режиме.

Для взаимодействия пользователя с программой программист должен предусмотреть какой-либо интерфейс. Консольные приложения, выполняемые в командной строке, используют текстовый интерфейс. Для создания текстового интерфейса принято использовать различные библиотеки, содержащие функции управления режимами экрана, работы с расширенными кодами клавиатуры, цветом, окнами и звуком.

8.1. Библиотека PDCurses

Среди Unix-систем как правило используется библиотека **ncurses**. Свободным аналогом этой библиотеки в среде Windows является библиотека **PDCurses**. Устанавливать её необходимо в ту же папку, в которую установлен компилятор GCC. Пакет программ, реализующих функции компилятора GCC для Windows называется MinGW и находится обычно в одной из папок: `c:\MinGW`, `c:\Program Files\MinGW` или `c:\Program Files\CodeBlocks\MinGW`. Для использования библиотеки PDCurses, необходимо включить в программу заголовочный файл библиотеки

```
#include <curses.h>
```

Тогда становятся доступными следующие типы данных, определённые в библиотеке:

- **chtype** — тип символов, с которыми работает PDCurses (он включает в себя код символа, цвет и дополнительные атрибуты);
- **bool** — логический тип (значение этого типа может быть ложным FALSE или истинным TRUE);
- **SCREEN** — структура, хранящая данные терминала (терминалом называют экран, на который осуществляется вывод; для Windows он совпадает с консольным окном командной строки, возможно раскрытым на весь экран комбинаций клавиш Alt+Enter);
- **WINDOW** — структура, хранящая данные окна (внутри терминала можно определять текстовые окна).

Программы на основе PDCurses используют для организации пользовательского интерфейса следующие понятия: экран (screen), окно (window) и подокно (sub-window). Экран представляет собой матрицу ячеек, в которые можно выводить символы. Окно — прямоугольная часть экрана. При использовании PDCurses программа обычно имеет следующую структуру:

```
...
#include <curses.h>
...
initscr();

работа с pdcurses

endwin();
...
```

Функция `initscr()` выполняет инициализацию данных, необходимых для работы библиотеки. При этом создаётся окно с именем `stdscr`, размер которого совпадает с размером всего экрана. После этого можно использовать функции PDCurses. При завершении работы с библиотекой нужно вызвать функцию `endwin()`. Рассмотрим пример

```
#include <curses.h>
int main(void)
{
// инициализация (должна быть выполнена
// перед использованием PDCurses)
initscr();
//перемещение курсора в стандартном экране y=10 x=30
move(10,30);
printw("Hello world !!!"); // вывод строки
refresh(); // обновить экран
getch(); // ждём нажатия символа
endwin(); // завершение работы с PDCurses
} //.
```

Здесь использованы следующие функции библиотеки:

- **move(y,x)** перемещает курсор в указанные координаты x и y экрана, начало координат находится в левом верхнем углу экрана, ось X направлена вправо, ось Y — вниз;
- **printw("Строка")** печатает строку в текущей позиции курсора;
- **refresh()** обновляет изменившуюся часть экрана.

Библиотека PDCurses позволяет ускорить вывод на экран за счёт того, что она использует запись не сразу на экран, а в буфер памяти, и когда нужно отобразить все изменения, используется функция `refresh()`.

Компиляция программы, использующей библиотеку **ncurses** в среде Unix-систем не вызывает трудностей. В Windows необходимо указать компоновщику место расположения файла pdcurses.a — откомпилированной версии библиотеки. Например, при работе в интегрированной среде разработки CodeBlocks, необходимо воспользоваться пунктом меню «Settings, Compiler and debugger» и на закладке «Linker settings» открывшегося диалогового окна в список «Link libraries» добавить путь к библиотеке (как правило C:\Program Files\CodeBlocks\MinGW\PDCurses\win32\pdcurses.a).

8.2. Функции ввода и вывода PDCurses

Стандартные функции ввода-вывода, описанные в библиотеках языка C (C++) нельзя использовать при программировании интерфейса PDCurses. Это связано с тем, что после инициализации PDCurses функцией `initscr()` вывод на экран должен происходить с использованием стандартного окна `stdscr`. Поэтому для ввода и вывода должны использоваться функции, объявленные в PDCurses.

8.2.1. Вывод на экран

Для вывода на экран в стандартное окно `stdscr` предусмотрены следующие функции:

- `printw()` — аналог функции `printf()`;
- `addch(chtype X)` — аналог функции putchar — выводит символ X в текущую позицию курсора и перемещает курсор на один символ вправо. Если курсор находился у правой границы экрана, то он перемещается в начало следующей строки.

- `insch(chtype X)` вставляет символ X слева от курсора, а все символы справа от курсора перемещаются на одну позицию вправо.
- `addchnstr(chtype *S, int n)` выводит первые n символов строки S начиная с текущей позиции курсора. Строка `chtype *S` представляет собой массив символов типа chtype. Если n=-1, то выводится вся строка.
- `addstr(char *S)` выводит строку S начиная с текущей позиции курсора. Строка S представляет собой массив символов типа char.
- `insnstr(char* S, int n)` вставляет первые n символов строки S в позицию, где расположен курсор, положение курсора не изменяется. Если n=-1, выводится вся строка.
- `insertln()` вставляет пустую строку в текущую позицию курсора.

Символы типа `chtype` отличаются от обычных символов типа `char` наличием у символа атрибутов мигания (A_BLINK), повышенной яркости (A_BOLD), нормального отображения (A_NORMAL), пониженной яркости (A_DIM), подчёркивания (A_UNDERLINE), инверсного изображения (A_REVERSE), а также информации о цвете. Например, операция

```
chtype C='a'|A_BOLD;
```

присваисает переменной C значение символа 'a' повышенной яркости.

8.2.2. Работа с цветом

Перед использованием цветов нужно проинициализировать палитру. Функция `has_colors()` позволяет прове-

рить, можно ли использовать цвета. Она возвращает истинное значение, если можно работать с цветом. Функция `start_color()` включает поддержку цвета. Далее необходимо назначить номера́ используемых цветов. Номер 0 зарезервирован для стандартного отображения. Номера начиная с 1 можно использовать в программе. Каждому номеру необходимо задать цвет символа и цвет фона из следующего списка:
COLOR_BLACK — черный,
COLOR_RED — красный,
COLOR_GREEN — зелёный,
COLOR_YELLOW — желтый,
COLOR_BLUE — синий,
COLOR_MAGENTA — малиновый,
COLOR_CYAN — голубой,
COLOR_WHITE — белый.
Для этого предусмотрена функция `init_pair()`. Первым её аргументом является номер цвета из палитры, вторым и третьим — наименования цветов символа и фона соответственно, например:

```
// Цвет с номером 1
init_pair(1, COLOR_RED, COLOR_WHITE);
// Цвет с номером 2
init_pair(2, COLOR_YELLOW, COLOR_BLUE);
и т.д.
```

Тем самым будет сформирована палитра цветов. Теперь можно использовать атрибуты символов, обозначающие цвет

```
chtype C='a'|COLOR_PAIR(1);  //.
```

Здесь переменной C присваивается значение символа 'a' с номером цвета 1 из палитры.

При выводе на экран строк или символов, для которых не указаны атрибуты используются атрибуты по умолчанию. Измененить атрибуты можно с помощью следующих функций.

Функция `int attron(int A)` включает атрибут A. По сути перечисленные выше названия цветов, а также атрибуты символов, являются константами целого типа, определёнными в библиотеке PDCurses. Поэтому атрибут A в прототипе функции имеет целый тип. Например, чтобы вывести на экран сообщение повышенной яркости цветом с номером 2 из палитры, устанавливаем два атрибута COLOR_PAIR(2), A_BOLD и выводим сообщение

```
attron(COLOR_PAIR(2));
attron(A_BOLD);
printw("Сообщение!!!");
```

Функция `int attroff(int A)` выключает атрибут A.

Функция `void bkgdset(int A)` устанавливает атрибуты с которыми очищается экран. Очистить экран можно с помощью функции `clear()`. При необходимости включения или выключения сразу нескольких атрибутов их необходимо перечислять, разделяя вертикальной чертой.

В качестве примера приведём программу вывода на экран красочного приветствия.

```
#include <curses.h>.
int main(void)
{
// инициализируем экран
    initscr();
// разрешаем работу с цветом
    start_color();
```

```
// инициализируем палитру
// добавляем цвет с номером 1
    init_pair(1, COLOR_RED, COLOR_GREEN);
// устанавливаем яркость символов
    attron(A_BOLD);
// устанавливаем цвет
    attron(COLOR_PAIR(1));
// перемещаем курсор
    move(10, 15);
// Выводим приветствие
    printw("Добро пожаловать!!!\n");
// выключаем яркость символов
    attroff(A_BOLD);
// включаем мерцание
    attron(A_BLINK);
// перемещаем курсор
    move(15, 15);
// Выводим строку
    printw("PDCurses");
// Обновляем экран
    refresh();
// Задержка до нажатия клавиши
    getch();
// выключаем PDCurses
    endwin();
}
```

8.2.3. Ввод с клавиатуры

Для ввода с клавиатуры предусмотрены следующие функции:

- `scanw()` — аналог функции scanf();

- `getch(void)` — аналог функции getchar() — возвращает введённый символ;
- `getstr(char *S)` — вводит строку в переменную S;
- `getnstr(char *S, int n)` — вводит строку в переменную S, но не длиннее n символов.

При построении пользовательского интерфейса довольно важнуь роль играет функция `getch()`. Зачастую интерфейс пользователя реализуется с виде меню, управляемого так называемыми клавишами управления. Эти клавиши существенно отличаются от алфавитно-цифровых. К ним относятся функциональные клавиши, клавиши управления курсором и множество других управляющих комбинаций, возникающих, например, при использовании модифицирующих клавиш Shift, Ctrl, Alt.

При нажатии управляющей клавиши в поток ввода поступает не один, а целая последовательность кодов клавиш. Используя библиотеку PDCurses, можно запрограммировать реакцию программы на нажатие любых клавиш, включая управляющие. Для этого во-первых необходимо перевести терминал в такой режим работы, когда вводимые символы сразу передаются программе без ожидания нажатия Enter. Это достигается вызовом функции `cbreak()`. Обратная ей функция `nocbreak()` возвращает прежние настройки. Во-вторых необходимо включить с помощью функции

```
int keypad(WINDOW *win, bool bf);
```

режим обработки программой управляющих клавиш. Первый аргумент этой функции указывает, для какого окна включается обработка управляющих клавиш. Пока нам известно только главное стандартное окно `stdscr`. Второй аргумент логического типа bool может принимать значения

TRUE (обработка включена) или FALSE (обработка выключена).

Последовательность кодов, введённую с клавиатуры можно интерпретировать как целое число, состоящее из нескольких байт. Поэтому результатом работы функции `getch()` является значение типа int, а не char. Коды символов, лежащие в диапазоне от 32 до 127, относятся к символам стандартной таблицы ASCII (см. Приложение). Коды от 128 до 255 представляют собой вторую половину таблицы символов и зависят от используемой кодовой страницы (для русского языка: CP866, CP1251 или KOI8-R). Остальные коды принадлежат управляющим клавишам и их комбинациям.

Для удобства в библиотеке PDCurses объявлены константы, позволяющие использовать осмысленные имена вместо кодов управлюящих клавиш. Далее перечислены некоторые их них:
KEY_DOWN = 0x102 — стрелка вниз,
KEY_UP = 0x103 — стрелка вверх,
KEY_LEFT = 0x104 — стрелка влево,
KEY_RIGHT = 0x105 — стрелка вправо,
KEY_HOME = 0x106 — Home,
KEY_BACKSPACE = 0x107 — Backspace,
KEY_DC = 0x14a — Delete,
KEY_IC = 0x14b — Insert,
KEY_ENTER = 0x157 — Enter,
KEY_END = 0x166 — End,
KEY_NPAGE = 0x152 — Page Down,
KEY_PPAGE = 0x153 — Page Up,
KEY_F(n) — функциональные клавиши, где n может принимать значения от 0 до 63. Полный перечень таких констант можно найти в файле `curses.h`.

Функция noecho() отключает автоматическое дублирование вводимых символов на экране, а функция echo() включает дублирование.

Функция halfdelay(int time) ограничивает время ожидания ввода символа временем time (измеряется в десятых долях секунды). Если в течение этого времени никакая клавиша не была нажата, функция getch() возращает значение ERR=-1. Отменить такой режим ввода можно с помощью функции nocbreak().

Рассмотрим несколько примеров. Ниже приведена программа, которая отображает на экране коды нажимаемых клавиш до тех пор, пока не нажата клавиша Esc (код Esc равен 27).

```
#include <curses.h>
int main(void)
{
// инициализация экрана
    initscr();
// разрешение работы с цветом
    start_color();
// добавление цвета с номером 1
    init_pair(1, COLOR_RED, COLOR_GREEN);
// заливка экрана зелёным цветом
    bkgdset(COLOR_PAIR(1));
    clear();
// включение обработки управляющих клавиш
    keypad(stdscr,TRUE);
// Обновление экрана
    refresh();
// Переменная для ввода
    int C;
do
```

```
{C=getch();      // Чтение с клавиатуры
 clear();        // Очистка экрана
 move(10,15);    // Перемещение курсора
 printw("%d",C); //Вывод кода
 refresh();}     //Обновление экрана
while (C!=27);   // Пока не Esc
// выключаем PDCurses
    endwin();
return 0;
}
```

8.2.4. Управление курсором

В предыдущих примерах была использована функция `move(y,x)` позиционирования курсора в ячейку экрана с указанными координатами. Такая последовательность задания координат (сначала y, затем x) обусловлена следующими обстоятельствами. С точки зрения PDCurses экран представляет собой прямоугольную матрицу ячеек, в которых могут отображаться символы. Положение элемента в матрице обычно определяется номером строки и номером столбца. Номер строки по смыслу совпадает с координатой `y`, а номер столбца — с координатой `x`.

Библиотека PDCurses содержит ещё несколько функций для работы с курсором.

Функция `getyx(WINDOW *win, int y, int x)` позволяет получить текущие координаты курсора относительно указанного окна win, например, относительно главного окна stdscr.

Функции

```
int getmaxx(WINDOW *win),
int getmaxy(WINDOW *win)
```

выдают максимальные значения координат курсора по горизонтали и вертикали относительно указанного окна.

Следующая программа позволяет перемещать по экрану курсор с помощью клавиш управления курсором (стрелки).

```
#include <curses.h>
int main(void)
{
// инициализация экрана
    initscr();
// включение обработки управляющих клавиш
    keypad(stdscr,TRUE);
// Обновление экрана
    refresh();
// Переменная для ввода
    int C;
// Текущие координаты курсора
    int x=10;
    int y=10;
    move(y,x);
// Максимальные координаты курсора
    int maxx=getmaxx(stdscr);
    int maxy=getmaxy(stdscr);
do
{C=getch(); // Чтение с клавиатуры
// Анализ нажатой клавиши
 if ((C==KEY_LEFT)&&(x>0)) x--;
 if ((C==KEY_RIGHT)&&(x<maxx)) x++;
 if ((C==KEY_UP)&&(y>0)) y--;
 if ((C==KEY_DOWN)&&(y<maxy)) y++;
 move(y,x);   // Перемещение курсора
 refresh();}    //Обновление экрана
```

```
while (C!=27);  // Пока не Esc
// выключение PDCurses
    endwin();
return 0;
}
```

8.3. Окна и панели PDCurses

В библиотеке PDCurses предусмотрена возможность работы с окнами в текстовом режиме. Окно представляет собой прямоугольную область экрана. На самом деле при работе с PDCurses всегда приходится работать с окнами, так как при инициализации библиотеки создаётся стандартное окно stdscr размером в экран. Описанные выше функции ввода-вывода, управления выводом и курсором предназначены для работы со стандартным окном и только с ним. Для работы с другими окнами необходимо использовать модифицированные аналоги этих функций. Модификация как правило заключается в том, что имя функции используется с приставкой «w» (например, wprintw вместо printw) и первым аргументом модифицированных функций является имя используемого окна.

8.3.1. Окна

Перечислим далее функции для работы с окнами.

```
WINDOW *newwin(int lines, int cols, int y, int x)
```

— создаёт окно с координатами левого верхнего угла (x,y), в котором lines строк и cols столбцов.

```
WINDOW *subwin(WINDOW *parent,
int lines, int cols, int y, int x)
```

— создаёт подокно с координатами левого верхнего угла (x,y) относительно всего экрана, в котором `lines` строк и `cols` столбцов, `parent` — родительское окно.

```
WINDOW *derwin(WINDOW *parent,
int lines, int cols, int y, int x)
```

— создаёт подокно в котором `lines` строк и `cols` столбцов с координатами левого верхнего угла (x,y) относительно родительского окна `parent`.

```
int delwin(WINDOW *win)
```

— удаляет окно/подокно `win`.

```
int mvwin(WINDOW *win, int y, int x)
```

— перемещает окно `win` в новую позицию (x,y) относительно экрана.

```
int mvderwin(WINDOW *win, int y, int x)
```

— перемещает подокно win в новую позицию (x,y) относительно родительского окна.

Проиллюстрируем использование окон примером. Модифицируем программу вывода кодов клавиш так, чтобы использовались окна. Начало программы традиционное для PDCurses

```
#include <curses.h>
int main(void)
{
// инициализация экрана
    initscr();
```

```
// разрешение работы с цветом
    start_color();
// добавление цвета с номером 1
    init_pair(1, COLOR_RED, COLOR_GREEN);
    init_pair(2, COLOR_RED, COLOR_BLUE);
// заливка экрана зелёным цветом
    bkgdset(COLOR_PAIR(1));
    clear();
```

Далее вычисляются размеры экрана — максимальные значения координат курсора

```
int maxx=getmaxx(stdscr);
int maxy=getmaxy(stdscr);
```

и создаётся окно в верхней части экрана для вывода заголовка программы

```
WINDOW *topwin=newwin(3,maxx,0,0); //.
```

Его высота составляет три строки, а ширина совпадает с шириной экрана. Окну `topwin` назначается цвет очистки с номером 2 и выполняется очистка окна,

```
wbkgdset(topwin,COLOR_PAIR(2));
wclear(topwin);
```

тем самым окно заливается синим цветом. Далее организуем вывод в окно `topwin`.

```
// Назначение атрибутов цвета вывода
wattron(topwin,COLOR_PAIR(2));
// Рисование  рамки окна
box(topwin,'|','-');
```

```
// Перемещение курсора в координаты 25,1
wmove(topwin,1,25);
// Вывод строки
wprintw(topwin,"Программа вывода кодов клавиш"); //.
```

Аналогично создаётся второе окно, в которое будут выводиться коды нажимаемых клавиш.

```
// Создание окна
WINDOW *mildwin=newwin(5,10,5,30);
// Назначение атрибутов цвета очистки
wbkgdset(mildwin,COLOR_PAIR(2));
// Очистка окна
wclear(mildwin);
// Назначение атрибутов цвета вывода
wattron(mildwin,COLOR_PAIR(2));
// Рисование  рамки окна
box(mildwin,'*','*');  //.
```

При выводе каких-либо данных в окно `mildwin` нарисованная звёздочками рамка может быть стёрта. Чтобы избежать этого создадим подокно немного меньшего размера внутри окна `mildwin`.

```
// Создание подокна
WINDOW *submildwin=derwin(mildwin,3,8,1,1);
// Назначение атрибутов цвета очистки
wbkgdset(submildwin,COLOR_PAIR(1));
// Очистка окна
wclear(submildwin);
// Назначение атрибутов цвета вывода
wattron(submildwin,COLOR_PAIR(1)); //.
```

Далее обновляется изображение во всех окнах и запускается цикл чтения с клавиатуры, пока не будет нажата клавиша

Esc (код 27). Код каждой нажатой клавиши выводится в подокно submildwin с пердварительной его очисткой.

```
// включение обработки управляющих клавиш
    keypad(submildwin,TRUE);
// Обновление экрана
    refresh();
// Обновление окна topwin
    wrefresh(topwin);
// Обновление окна mildwin
    wrefresh(mildwin);
// Обновление подокна submildwin
    wrefresh(submildwin);
// Переменная для ввода
    int C;
do
{ // Чтение с клавиатуры из окна submildwin
C=wgetch(submildwin);
// Очистка окна  submildwin
 wclear(submildwin);
 // Перемещение курсора
 wmove(submildwin,1,2);
 //Вывод кода клавиши
 wprintw(submildwin,"%d",C);
 // Обновление окна  submildwin
 wrefresh(submildwin);
}
 while (C!=27); // Пока не Esc
// выключение PDCurses
    endwin();
return 0;  //.
```

Окна в PDCurses находятся в одной плоскости экрана и при наложении могут стирать содержимое друг друга. Для устранения этого недостатка в PDCurses существует понятие панели.

8.3.2. Панели

Панель представляет собой окно, обладающее свойством глубины. Глубина указывает в каком порядке располагаются панели на экране. Каждую панель можно переместить вверх или вниз относительно других панелей, а также скрыть и показать. Для использования панелей нужно включить в программу заголовочный файл `panel.h`. Панель создается на основе уже существующего окна с помощью функции

```
PANEL *new_panel(WINDOW *win)
```

и помещается в вершину стека панелей, то есть на верхний слой, например,

```
PANEL *panel1;
panel1 = new_panel(topwin);
```

Перечислим основные функции для работы с панелями, реализованные в PDCurses:

```
int del_panel(PANEL *panel1)
```

— удаляет панель panel1. Окно, связанное с панелью, не удаляется. При необходимости его нужно удалить отдельно.

```
WINDOW *panel_window(const PANEL *panel1)
```

— возвращает указатель на окно связанное с панелью panel1.

```
int hide_panel(PANEL *panel1)
```

— скрывает панель. При этом панель удаляется из стека панелей.

`int show_panel(PANEL *panel1)`

— показывет скрытую панель. При этом панель помещается в стек панелей на самый верхний уровень.

`int top_panel(PANEL *panel1)`

— перемещает панель на самый верх, поверх остальных панелей.

`int bottom_panel(PANEL *panel1)`

— перемещает панель ниже всех панелей.

`int move_panel(PANEL *pannel1, int y, int x)`

— перемещает панель так, чтобы её верхний левый угол оказался в точке (x,y).

`int replace_panel(PANEL *panel1, WINDOW *win)`

— меняет в панели panel1 текущее окно на другое окно win.

`PANEL *panel_above(const PANEL *pan)`

— возвращает указатель на панель, которая находится выше на один уровень. Если такой панели нет, то возвращает 0.

`PANEL *panel_below(const PANEL *pan)`

— возвращает указатель на панель, которая находится ниже на один уровень. Если такой панели нет, то возвращает 0.

`void update_panels()`

— обновляет стек панелей. Для отображения изменений на экране нужно вызвать функцию `doupdate()`.

8.4. Работа с мышью в PDCurses

В библиотеке PDCurses работа с мышью реализована следующим образом. Функция `mousemask()`, позволяет включить или выключить отслеживание событий от мыши. Её первым аргументом является так называемая битовая маска, в которой указывается, какие события необходимо отслеживать. Для основных комбинаций битовых масок предусмотрены константы, например,
BUTTON1_CLICKED — щелчок левой кнопкой мыши,
BUTTON1_DOUBLE_CLICKED — двойной щелчок левой кнопкой мыши,
BUTTON1_TRIPLE_CLICKED — тройной щелчок левой кнопкой мыши,
ALL_MOUSE_EVENTS — отслеживание всех событий от мыши.

Вторым аргументом функции `mousemask()` может являться либо указатель на вспомогательную переменную, предназначенную для временного хранения предыдущей битовой маски, либо константа NULL.

Обнаружение события, сгенерированого мышью, равносильно обнаружению в стандарном потоке ввода кода KEY_MOUSE = 539. Такой подход позволяет работу с мышью объединить в один цикл с обработкой нажатий клавиш клавиатуры. Следующий цикл будет выполняться пока не произойдёт любое событие от мыши

```
int C;
keypad(stdscr,TRUE);
do
C=getch();
while(C!=KEY_MOUSE);
```

Вместо стандартного окна `stdscr` здесь можно использовать любое другое окно или панель.

После обнаружения события мыши его необходимо считать из очереди. Само событие представляет собой структуру данных, в которой указано, какое именно событие произошло, координаты мыши при его наступлении и т.д. В PDCurses предусмотрен специальный тип данных MEVENT, а также функция `nc_getmouse(MEVENT *)` считывающая события из очереди. Структура MEVENT представляет собой объединение разнотипных переменных, среди которых присутствуют переменные x и y, содержащие экранные координаты указателя мыши во время наступления события. Воспользоваться ими можно с помощью уточнённого имени: ИмяСтруктуры.x, ИмяСтруктуры.y. Следующий фрагмент программы отображает в верхнем левом углу экрана координаты щелчков мыши

```
int C;
keypad(stdscr,TRUE);
do
{
move(0,0);
C=getch();
if (C==KEY_MOUSE)
  {
   clear();
   MEVENT M;
   nc_getmouse(&M);
   printw("%d   %d",M.x,M.y);
  }
}
while(C!=27);
```

Пример.

Пусть необходимо написать программу, реализующую переключение активного окна с помощью мыши.

В верхней части экрана расположим заголовочное окно, в котором будет отображаться номер активного окна и координаты щелчка мышью. В средней части экрана будут находиться три окна, одно из которых может быть активным. Признаком его активности выберем красный цвет фона. Щелчок мыши на любом из трёх окон должен делать его активным. Щелчок правее окон снимает активность со всех окон.

Поскольку действия с окнами средней части экрана являются однотипными, целесообразно описать массив из трёх окон, а именно

```
// Описываем массив окон
WINDOW *win[3];  //.
```

Вначале главной функции `main` инициализируется экран, работа с цветом, назначаются номера цветов палитры

```
// инициализация экрана
    initscr();
// разешение работы с цветом
    start_color();
// добавление цвета с номером 1, 2, 3
    init_pair(1, COLOR_WHITE, COLOR_GREEN);
    init_pair(2, COLOR_WHITE, COLOR_BLUE);
    init_pair(3, COLOR_WHITE, COLOR_RED);   //.
```

Цвет с номером 1 будем использовать для заливки всего экрана, цвет с номером 2 — для заливки неактивного окна, а цвет с номером 3 — для заливки активного окна. Далее очищаем весь экран, вычисляем максимальные значения координат курсора

```
// заливка экрана зелёным цветом
    bkgdset(COLOR_PAIR(1));
    clear();
// вычисление максимальных координат
    int maxx=getmaxx(stdscr);
    int maxy=getmaxy(stdscr);
```

и создаем заголовочное окно

```
// создание окна
WINDOW *topwin=newwin(3,maxx,0,0);
// назначение цвета очистки
wbkgdset(topwin,COLOR_PAIR(2));
// назначение атрибутов вывода
wattron(topwin,COLOR_PAIR(2));
//очистка окна
wclear(topwin);
//рисование рамки окна
box(topwin,'|','-');
// перемещение курсора в координаты 25,1
wmove(topwin,1,25);
// вывод сообщения
wprintw(topwin,"No click!!!");  //.
```

Здесь создано окно с именем `topwin` размером 3 строки и `maxx` столбцов с координатами левого верхнего угла (0,0). Цвет с номером 2 палитры (белый на синем) назначен для очистки и для вывода в это окно. После очистки окна нарисована рамка и выведен текст «No click!!!».

Следующий фрагмент создаёт окна, принадлежащие массиву win.

```
int k;   // переменная цикла
for(k=0;k<=2;k++)
```

```
{
// создание окна с номером k
win[k]=newwin(10,25,5,k*25+k);
wbkgdset(win[k],COLOR_PAIR(2));
wclear(win[k]);
wattron(win[k],COLOR_PAIR(2));
box(win[k],'|','-');
}  //.
```

Каждое из них имеет размер 10 строк и 25 столбцов. Окна расположены горизонтально, поэтому координата y левого верхнего угла равна 5, а координата x вычисляется по формуле $x = k * 25 + k$, где k — номер окна. Окна заливаются синим цветом и прорисовываются рамки. Далее выполняется обновление экрана, очистка всех окон, включение реакции программы на щелчки мышью и нажатия функциональных клавиш,

```
// Обновление экрана
   refresh();
// Обновление окна topwin
   wrefresh(topwin);
// Обновление окон массива win
   for(k=0;k<=2;k++) {wrefresh(win[k]);}
// Включение обработки событий мыши
   mousemask(ALL_MOUSE_EVENTS,NULL);
// Включение обработки функциональных клавиш
   keypad(stdscr,TRUE);
```

а также объявляются целые переменные `C` и `active` для хранения кода нажатой клавиши и номера активного окна соответственно:

```
int C;
int active=0;  //.
```

Следующая часть программы представляет собой цикл обработки сообщений от мыши:

```
// начало цикла пока не нажата клавиша Esc
do
{
// Считывание кода нажатой клавиши
C=getch();
// если есть события от мыши
if (C==KEY_MOUSE)
  {
  // Считывание события из очереди
   MEVENT M;
   nc_getmouse(&M);
   // Анализ координат нажатия
   if ((M.y>=5)&&(M.y<15)) {active=M.x/25;}
   // Вывод номера активного окна
   wmove(topwin,1,25);
   wprintw(topwin,"Active%d %d %d",active,M.x,M.y);
   wrefresh(topwin);
   // Перерисовка окон
   repaint(active);
  }
}
while(C!=27);  //.
```

Если щелчок мыши произошёл в диапазоне вертикальных координат $5 \leq y < 15$, то меняется активное окно. Номер активного окна является результатом целочисленного деления координаты нажатия на размер окон по оси OX. Например, щелчок в позиции с координатами $y = 10$, $x = 38$ делает активным среднее окно, так как $38/25 = 1$.

При смене активного окна нужно выполнить перери-

совку окон из массива win. Для этого предусмотрим функцию `repaint(int a)`. В качестве параметра функция принимает номер активного окна `a` и перерисовывает все окна. Перерисовка активного и неактивных окон отличается только номером используемого цвета.

```
void repaint(int a)
{
int k; // переменная цикла
// цикл по окнам
for(k=0;k<=2;k++)
{
  if (k==a) // если окно активное
  {
    // то назначается цвет 3
    wbkgdset(win[k],COLOR_PAIR(3));
    //очистка окна
    wclear(win[k]);
    // рисование рамки
    box(win[k],'|','-');
    // обновление окна
    wrefresh(win[k]);
  }
  else
  {
    // иначе назначается цвет 2
    wbkgdset(win[k],COLOR_PAIR(2));
    //очистка окна
    wclear(win[k]);
    // рисование рамки
    box(win[k],'|','-');
    // обновление окна
    wrefresh(win[k]);
```

```
  }
}
}  //.
```

Переменная цикла k поочерёдно принимает значения номера перерисовываемого окна. Если этот номер совпадает с номером активного окна, то происходит перерисовка с номером цвета 3. Все остальные окна перерисовываются с номером цвета 2. Ниже приведён полный текст рассмотренной программы.

```
#include <curses.h>

// массив окон
WINDOW *win[3];

// функция перерисовки окон
void repaint(int a)
{
 int k; // переменная цикла
 // цикл по окнам
 for(k=0;k<=2;k++)
   {
   if (k==a) // если активное окно
     {
     // то назначается цвет 3
     wbkgdset(win[k],COLOR_PAIR(3));
     //очистка окна
     wclear(win[k]);
     // рисование рамки
     box(win[k],'|','-');
     // обновление окна
     wrefresh(win[k]);
```

```
      }
    else
      {
      // иначе назначается цвет 2
      wbkgdset(win[k],COLOR_PAIR(2));
      //очистка окна
      wclear(win[k]);
      // рисование рамки
      box(win[k],'|','-');
      // обновление окна
      wrefresh(win[k]);
      }
    }
 }

// Главная функция
int main(void)
{
    initscr();  // инициализация экрана
    start_color(); // разешение работы с цветом

// добавление цветов в палитру
    init_pair(1, COLOR_WHITE, COLOR_GREEN);
    init_pair(2, COLOR_WHITE, COLOR_BLUE);
    init_pair(3, COLOR_WHITE, COLOR_RED);

// заливка экрана зелёным цветом
    bkgdset(COLOR_PAIR(1));
    clear();
    int maxx=getmaxx(stdscr);
    int maxy=getmaxy(stdscr);
```

```
// Создание заголовочного окна
WINDOW *topwin=newwin(3,maxx,0,0);
wbkgdset(topwin,COLOR_PAIR(2));
wclear(topwin);
wattron(topwin,COLOR_PAIR(2));
box(topwin,'|','-');
wmove(topwin,1,25);
wprintw(topwin,"No click!!!");

// Создание окон массива win
int k;
for(k=0;k<=2;k++)
{
win[k]=newwin(10,25,5,k*25+k);
wbkgdset(win[k],COLOR_PAIR(2));
wclear(win[k]);
wattron(win[k],COLOR_PAIR(2));
box(win[k],'|','-');
}

// Обновление экрана и окон
   refresh();
   wrefresh(topwin);
   for(k=0;k<=2;k++) {wrefresh(win[k]);}
// Включение обработки событий мыши
   mousemask(ALL_MOUSE_EVENTS,NULL);
// Включение обработки функциональных клавиш
   keypad(stdscr,TRUE);

int C; // Переменная для ввода
int active;
// Цикл обработки событий
```

```
do
{
C=getch();
if (C==KEY_MOUSE)
  {
   MEVENT M;
   nc_getmouse(&M);
   if ((M.y>=5)&&(M.y<15)) {active=M.x/25;}
   wmove(topwin,1,25);
   wprintw(topwin,"Active%d %d %d",active,M.x,M.y);
   wrefresh(topwin);
   repaint(active);
  }
}
while(C!=27);
    endwin();
return 0;
}
```

8.5. Практическое задание

При выполнении практического задания придерживайтесь следующей последовательности действий.

1. **Обязательно** изучите описание, предлагаемое выше.
2. Разберите примеры, приводимые в описании.
3. Осмыслите задачу, предлагаемую в вашем варианте.
4. Составьте словесный алгоритм решения задачи.
5. Представьте алгоритм решения в виде блок-схемы.
6. Переведите алгоритм на язык программирования.

8.5.1. Варианты заданий

Вариант 1.

Реализуйте на экране горизонтальное меню, управляемое клавишами управления курсором. При выборе пунктов меню должны происходить какие-нибудь действия.

Вариант 2.

Реализуйте на экране вертикальное меню, управляемое клавишами управления курсором. При выборе пунктов меню должны происходить какие-нибудь действия.

Вариант 3.

Реализуйте на экране меню, управляемое мышью. При выборе пунктов меню должны происходить какие-нибудь действия.

Вариант 4.

Реализуйте на экране двухуровневое меню, управляемое клавишами управления курсором. При выборе пунктов меню должны происходить какие-нибудь действия.

Вариант 5.

Реализуйте перемещение по экрану прямоугольника, управляемое клавишами управления курсором, с возможностью приостановки движения и его продолжения.

Вариант 6.

Напишите программу решения системы трёх линейных уравнений. Предусмотрите интерфейс пользователя.

Вариант 7.

Осуществите просмотр в окне меньшего, чем экран размера текстового файла с возможностью прокрутки текста.

Вариант 8.

В текстовом режиме экрана реализуйте возможность рисования (какими-либо символами) прямоугольников с помощью клавиш управления курсором.

Вариант 9.

В текстовом режиме экрана реализуйте возможность рисования (какими-либо символами) линий с помощью клавиш управления курсором.

Вариант 10.

В текстовом режиме экрана реализуйте возможность рисования (какими-либо символами) прямоугольников с помощью мыши.

Вариант 11.

В текстовом режиме нарисованный символами объект (прямоугольник, линия) должны перемещаться под управлением клавиш управления курсором.

Вариант 12.

На экране присутствуют два текстовых окна. В одно из них загружается текстовый файл. В другом — при нажатии выбранной клавиши отображается тот же текст, зашифрованный азбукой Морзе.

Вариант 13.

На экране присутствуют два текстовых окна. В одно из них выводится содержимое текстового файла. В другом — при нажатии выбранной клавиши отображается тот же текст, но зашифрованный. Например, к коду каждого символа добавляется константа.

Вариант 14.

Напишите программу расчёта определителя матрицы 3x3. Предусмотрите интерфейс пользователя.

Вариант 15.

Пользователь набирает в отведённом для этого окне текстовый файл. При нажатии F12 текст записывается в файл.

9. ЛАБОРАТОРНАЯ РАБОТА №5 ИНТЕРПОЛИРОВАНИЕ АЛГЕБРАИЧЕСКИМИ МНОГОЧЛЕНАМИ

Цели работы:

- ознакомление с постановкой задачи интерполяции;
- ознакомление с графической библиотекой;
- приобретение навыков построения графиков.

9.1. Введение в задачу интерполяции

Пусть между некими величинами x и y существует функциональная зависимость $y = f(x)$, аналитический вид которой неизвестен. В результате эксперимента получены пары значений (x_k, f_k), $f_k = f(x_k)$, $k = \overline{0, n}$, $n + 1$ — количество выполненных измерений. Тем самым функция $f(x)$ задана таблицей значений в некоторых точках (узлах интерполяции) отрезка $[a, b]$.

Задача интерполяции заключается в подборе на основе экспериментальных данных функции $g(x)$ в некотором смысле близкой к $f(x)$. Будем искать функцию $g(x)$ в классе алгебраических многочленов вида

$$g(x) = P_n(x) = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n. \qquad (1)$$

Очевидно функция $P_n(x)$ полностью определяется коэффициентами a_k, $k = \overline{0, n}$. Потребуем, чтобы значения многочлена $P_n(x)$ совпадали с измеренными значениями функции в узлах интерполяции, в результате получим систему линей-

ных уравнений относительно a_k:

$$\begin{cases} a_0 + a_1 x_0 + a_2 x_0^2 + ... + a_n x_0^n = f_0, \\ a_0 + a_1 x_1 + a_2 x_1^2 + ... + a_n x_1^n = f_1, \\ ... \\ a_0 + a_1 x_k + a_2 x_k^2 + ... + a_n x_k^n = f_k, \\ ... \\ a_0 + a_1 x_n + a_2 x_n^2 + ... + a_n x_n^n = f_n. \end{cases} \tag{2}$$

Решение системы уравнений (2) позволяет выразить коэффициенты a_k через экспериментальные данные (x_k, f_k), подставив которые в формулу (1), можно получить выражение для интерполяционного многочлена. Система уравнений (2) имеет единственное решение, которое можно записать различным образом. Широко распространены интерполяционные многочлены в форме Лагранжа и Ньютона.

Интерполяционная формула Лагранжа позволяет представить многочлен (1) в виде

$$P_n(x) = \sum_{k=0}^{n} f_k \frac{(x-x_0)...(x-x_{k-1})(x-x_{k+1})...(x-x_n)}{(x_k-x_0)...(x_k-x_{k-1})(x_k-x_{k+1})...(x_k-x_n)} = \\ = \sum_{k=0}^{n} f_k \frac{\prod\limits_{j=0, j\neq k}^{n} (x-x_j)}{\prod\limits_{j=0, j\neq k}^{n} (x_k-x_j)}.$$

Интерполирование по формуле Лагранжа довольно просто реализуется в виде подпрограммы, но имеет существенный недостаток: при добавлении новых узлов интерполяционный многочлен нужно строить заново.

Преодолеть указанный недостаток позволяет использование интерполяционной формулы Ньютона. Она выражает

многочлен $P_n(x)$ через значение $f(x)$ в одном из узлов и так называемые разделённые разности.

Для построения разделённых разностей будем считать, что среди точек x_k нет совпадающих. Разделёнными разностями первого порядка называются величины

$$f_{i,j} = \frac{f_j - f_i}{x_j - x_i}, \qquad i, j = \overline{0, n}, \quad i \neq j.$$

На основе разделённых разностей первого порядка, составленным по соседним узлам $f_{0,1}$, $f_{1,2}$,..., $f_{n-1,n}$, можно построить разделённые разности второго порядка

$$f_{0,1,2} = \frac{f_{1,2} - f_{0,1}}{x_2 - x_0},$$

$$f_{1,2,3} = \frac{f_{2,3} - f_{1,2}}{x_3 - x_1},$$

$$\ldots$$

$$f_{n-2,n-1,n} = \frac{f_{n-1,n} - f_{n-2,n-1}}{x_n - x_{n-2}}.$$

Аналогично определяются разделённые разности более высокого $(k+1)$-ого порядка по уже известным разностям порядка k:

$$f_{j,j+1,\ldots,j+k,j+k+1} = \frac{f_{j+1,j+2,\ldots,j+k+1} - f_{j,j+1,j+k}}{x_{j+k+1} - x_j}.$$

Использование разделённых разностей позволяет записать интерполяционный многочлен Ньютона для интерполирования вперед (начальной точкой выбрана x_0)

$$\begin{gathered} P_n(x) = f_0 + f_{0,1}(x - x_0) + \\ + f_{0,1,2}(x - x_0)(x - x_1) + \ldots + \\ + f_{0,1,\ldots,n-1,n}(x - x_0)(x - x_1)\ldots(x - x_{n-1}) \end{gathered} \tag{3}$$

и для интерполирования назад (начальной точкой выбрана x_n)

$$P_n(x) = f_n + f_{n-1,n}(x - x_n) + \\ +f_{n-2,n-1,n}(x - x_n)(x - x_{n-1}) + ... + \\ +f_{0,1,...,n-1,n}(x - x_n)(x - x_{n-1})...(x - x_1). \quad (4)$$

Нумерация узлов при проведении расчётов по формуле Ньютона не имеет значения. Это позволяет легко добавлять новые узлы. Например, добавление одного узла x_{n+1} приводит к появлению в интерполяционном многочлене (3) слагаемого вида

$$f_{0,1,...,n,n+1}(x - x_0)(x - x_1)...(x - x_{n-1})(x - x_n).$$

Рассмотрим теперь один из способов расчёта коэффициентов многочлена (3) в программе. Заметим, что коэффициентами многочлена являются одно из значений функции в начальном узле $f(x_0)$, одна из разделённых разностей первого порядка $f_{0,1}$, одна из разделённых разностей второго порядка $f_{0,1,2}$ и т.д. Следовательно, остальные значения функции и разделённых разностей являются промежуточными данными. Пусть значения x_k и f_k расположены в одномерных массивах из n+1 элементов, изображенных для иллюстрации в первой и второй строке табл. 9.

Начнем вычислять разделённые разности с конца массива. После вычисления величины $f_{n-1,n} = (f_n - f_{n-1})/(x_n - x_{n-1})$ значение f_n, занимающее последнее место в массиве больше не потребуется. Поэтому вычисленную конечную разность можно сохранить на последнем месте в массиве f, как показано в строке 3 таблицы 9. Аналогично после вычисления величины $f_{n-2,n-1} = (f_{n-1} - f_{n-2})/(x_{n-1} - x_{n-2})$ не потребуется предпоследний элемент массива f_{n-1}. Поэтому на его месте можно сохранить вычисленную разность (стро-

ка 4) и т.д. В результате вычислния всех разделённых разностей первого порядка

```
for(k=n;k>=1;k--)
   {f[k]=(f[k]-f[k-1])/(x[k]-x[k-1]);}
```

содержимое массива f будет соответствовать строке 5 табл. 9.

Таблица 9

Вычислений конечных разностей

1	x_0	x_1	x_2	...	x_{n-1}	x_n
2	f_0	f_1	f_2	...	f_{n-1}	f_n
3	f_0	f_1	f_2	...	f_{n-1}	$f_{n-1,n}$
4	f_0	f_1	f_2	...	$f_{n-2,n-1}$	$f_{n-1,n}$
...						
5	f_0	$f_{0,1}$	$f_{1,2}$	...	$f_{n-2,n-1}$	$f_{n-1,n}$
6	f_0	$f_{0,1}$	$f_{0,1,2}$	...	$f_{n-2,n-1}$	$f_{n-2,n-1,n}$
...						
7	f_0	$f_{0,1}$	$f_{0,1,2}$	...	$f_{0,1,...,n-1}$	$f_{0,1,...,n-1,n}$

Разделённые разности второго порядка вычисляются на основе разделенных разностей первого порядка и хранятся в элементах массива f[2]...f[n]:

```
for(k=n;k>=2;k--)
   {f[k]=(f[k]-f[k-1])/(x[k]-x[k-2]);}  //.
```

Разделённые разности порядка j вычисляются на основе разделенных разностей порядка j-1 и хранятся в элементах массива f[j]...f[n]:

```
for(k=n;k>=j;k--)
   {f[k]=(f[k]-f[k-1])/(x[k]-x[k-j]);}  //.
```

В результате массив `f` будет содержать коэффициенты интерполяционного многочлена Ньютона, а все промежуточные данные будут утеряны (строка 7).

Вычисление значений интерполяционного многочлена наиболее экономично проводить по схеме Горнера, получаемой путем последовательного вынесения за скобки множителей $(x - x_i)$

$$P_n(x) = f_0 + (x - x_0)[f_{0,1} + (x - x_1)[f_{0,1,2} + ...+ \\ +(x - x_{n-2})[f_{0,1,...,n-1} + (x - x_{n-1})f_{0,1,...,n}]]...].$$

9.2. Графическая библиотека WinBGIm

В графическом режиме область вывода на экран (это может быть весь экран монитора или графическое окно) представляет собой прямоугольную матрицу точек, цвет каждой из которых можно изменять отдельно.

Для эффективного программирования графического режима целесообразно использовать различные графические библиотеки. Одной из них является библиотека WinBGIm — аналог WinBGI фирмы Borland. Эта библиотека содержит огромное число графических процедур и функций. Предусмотреть запуск графического режима можно в любом месте программы. Для этого нужно включить в программу заголовичный файл `graphics.h` и описать две вспомогательные переменные целого типа

```
#include <graphics.h>
...
    int gd, gm;  //.
```

Переменная `gd` (graphic driver) используется для хранения номера (кода) графического драйвера, а `gm` (graphic mode)

— графического режима. Имена переменных `gd` и `gm` могут быть любыми. Перед инициализацией графики необходимо присвоить значения переменным `gd` и `gm`. Однако можно предоставить выбор наилучшего драйвера и режима функции автоопределения параметров графики

```
void detectgraph(int *graphdriver, int *graphmode);
```

Возможные значения номера графического драйвера приведены в табл. 10. Чаще всего на персональных компьютерах используется графический драйвер VGA с номером 9 с одним из трёх возможных режимов, перечисленных в табл. 11.

Таблица 10

Графические драйверы

Название	**Номер**
DETECT	0 (автоопределение)
CGA	1
MCGA	2
EGA	3
ECGA64	4
EGAMONO	5
IBM8514	6
HERCMONO	7
ATT400	8
VGA	9
PC3270	10

Таблица 11

Режимы VGA

Название	**Номер**	**Разрешение**	**Палитра**
VGALO	0	640x200	16 цветов
VGAMED	1	640x350	16 цветов
VGAHI	2	640x480	16 цветов

Инициализацию графического режима выполняет функция

```
void initgraph(int *graphdriver,
     int *graphmode, char *path);
```

Здесь параметр `path` — путь к файлу графического драйвера. При автоопределении параметров в качестве параметра `path` целесообразно использовать пустую строку. При успешном выполнении функции `initgraph` будет создано дополнительное окно графического вывода программы. По окончании работы с графическим режимом его необходимо деинициализировать (закрыть) с помощью функции

```
void closegraph(int w=ALL_WINDOWS); //.
```

Поскольку имеется возможность открытия нескольких графических окон, параметр функции w позволяет указать, какое именно графическое окно необходимо закрыть. Этот параметр может принимать только одно из двух значений: CURRENT_WINDOW (закрытие текущего окна) и ALL_WINDOWS (закрытие всех графических окон, используется по умолчанию).

В следующем примере выполняется инициализация графического режима и его закрытие после нажатия любой клавиши.

```
#include <iostream.h>
#include <graphics.h>

int main()
{
    int gd, gm;
    detectgraph(&gd,&gm);
    initgraph(&gd,&gm,"");
    getch();
    closegraph();
    return 0;
}  //.
```

Для кодирования цвета в библиотеке WinBGIm используется нумерация цветов целыми числами. Предусмотрены также символические названия цветов, приведённые в табл. 12.

9.2.1. Графический курсор, указатель

Процесс построения изображений в графическом окне опирается на систему координат с началом в левом верхнем углу окна. Ось X направлена вправо, ось Y — вниз. Аналогом текстового курсора в графическом режиме является невидимый текущий указатель. Зачастую рисование объектов выполняется относительно текущего указателя. Максимальное значение координат указателя зависит от разрешения (количества точек, размещаемых в окне). Получить максимально возможные координаты можно с помощью функций

```
int getmaxx(void);
int getmaxy(void);  //.
```

Таблица 12

Символические названия цветов

Название	**Номер**
BLACK	0
BLUE	1
GREEN	2
CYAN	3
RED	4
MAGENTA	5
BROWN	6
LIGHTGRAY	7
DARKGRAY	8
LIGHTBLUE	9
LIGHTGREEN	10
LIGHTCYAN	11
LIGHTRED	12
LIGHTMAGENTA	13
YELLOW	14
WHITE	15

Перемещение указателя без прорисовки на экране осуществляется с помощью одной из функций

```
void moveto(int x, int y);
void moverel(int dx, int dy);  //.
```

Первая из них перемещает указатель в указанные координаты, вторая смещает указатель на величины dx и dy относительно текущей позиции.

Узнать текущие координаты указателя можно с помощью функций

```
int getx(void);
int gety(void);  //.
```

9.2.2. Рисование точек и линий

В библиотеке WinBGIm вывод точки осуществляет функция

```
void putpixel(int x, int y, int color);
```

где x, y — координаты расположения точки, color — цвет. Например, команда

```
putpixel(getmaxx()/2, getmaxy()/2, GREEN);
```

выводит в центре экрана зеленую точку.

Для построение отрезков прямых служат функции `line`, `lineto`, `linerel`. Функция

```
void line(int x1, int y1, int x2, int y2);
```

рисует линию с началом в точке с координатами (x1, y1) и концом в точке (x2,y2). Следует заметить, что многие функции рисования графических примитивов, в частности функция `line`, не имеют параметра установки цвета. В этом случае цвет задается перед рисованием объекта функцией

```
void setcolor (int color);
```

например,

```
setcolor(RED);
line(0,0,100,200);  //.
```

Аналогично функция `setbkcolor (int color)` назначает текущий цвет фона.

Функции

```
int getcolor(void);
int getbkcolor(void);
```

позволяют получить номер текущего цвета указателя и фона соответсвенно, а функция

```
unsigned getpixel(int x, int y);
```

— номер цвета точки окна с указанными координатами.

Функция

```
void lineto(int x, int y);
```

строит линию из точки текущего положения указателя в точку с координатами x,y.

Функция

```
void linerel(int dx, int dy);
```

строит линию между точкой текущего положения указателя (x, y) и точкой (x+dx, y+dy).

Допускается установка стиля линии функцией

```
void setlinestyle (int A, unsigned B, int C);
```

где параметр A определяет тип линии, возможные значения которого описаны в виде констант и приведены в табл. 13.

Таблица 13

Стили линий

Название	**Номер**	**Описание**
SOLID_LINE	0	сплошная
DOTTED_LINE	1	пунктирная
CENTER_LINE	2	штрих-пунктирная
DASHED_LINE	3	штриховая
USERBIT_LINE	4	стиль, задаваемый образцом

Параметр C определяет толщину линии. Возможные значение толщины:
NORM_WIDTH=1 (линия в 1 пиксел) и
THICK_WIDTH=3 (жирная линия толщиной в 3 пиксела).

Параметр B является образцом стиля и используется, если стиль линии равен USERBIT_LINE (в протипном случае B=0). Образец задаётся в формате двухбайтового числа. В его двоичном представлении единицы соответствуют светлым участкам линии, нули — тёмным. Образец штриховой линии может выглядеть как

```
1111000011110000=0xF0F0,
```

а пунктирной линии как

```
1010101010101010=0xAAAA,
```

например,

```
setlinestyle(SOLID_LINE,0,NORM_WIDTH);
line(30,30, 555,400);
```

или

```
setlinestyle(USERBIT_LINE,0x9898,THICK_WIDTH);
line(30,30, 555,400);  //.
```

9.2.3. Рисование прямоугольников

Прямоугольник можно построить, указав координаты его двух характерных точек, например левого верхнего (x1, y1) и правого нижнего (x2, y2) углов.

Функция

```
void rectangle(int x1, int y1, int x2, int y2);
```

рисует незакрашенный прямоугольник, а функция

```
void bar(int x1, int y1, int x2, int y2);
```

— закрашенный прямоугольник.

Функция

```
void bar3d(int x1, int y1,
           int x2, int y2, int d, int b);
```

вычерчивает трёхмерный закрашенный прямоугольник (параллелепипед). Параметр `d` определяет глубину трёхмерного контура, а параметр `b` — закрыта ли верхняя «крышка» параллелепипеда.

9.2.4. Рисование окружностей и эллипсов

Построение эллиптических дуг осуществляет функция

```
void ellipse(int x, int y, int a, int b,
       int Rx, int Ry);
```

где `x`, `y` — координаты центра эллипса, `a, b` — начальный и конечный углы, отсчитанные от горизонтали, Rx, Ry — радиусы в горизонтальном и вертикальном направлениях. Для построения замкнутого эллипса начальный и конечный углы устанавливают равными 0 и 360 градусов. Окружность является частным случаем эллипса при одинаковых горизонтальном и вертикальном радиусах. Так, например, оператор

```
ellipse(100, 100, 0, 360, 50,50);
```

рисует в графическом окне окружность.

Функция

```
void fillellipse(int x, int y, int Rx, int Ry);
```

рисует закрашенный эллипс.

Стиль и цвет заливки назначаются функцией

```
void setfillstyle(int style, int color);
```

где `style` — стиль заливки, `color` — цвет заливки. Возможные значения стиля заливки приведены в табл. 14.

Таблица 14

Стили заливки

Название	**Номер**	**Описание**
EMPTY_FILL	0	сплошная цветом фона
SOLID_FILL	1	сплошная
линиями		
LINE_FILL	2	горизонтальными
LTSLASH_FILL	3	наклонными
SLASH_FILL	4	толстыми наклонными
BKSLASH_FILL	5	толстыми наклонными
LTBKSLASH_FILL	6	наклонными
клеткой		
HATCH_FILL	7	тонкими линиями
XHATCH_FILL	8	толстыми линиями
INTERLEAVE_FILL	9	диагональной штриховкой
WIDE_DOT_FILL	10	редкими точками
CLOSE_DOT_FILL	11	частыми точками
USER_FILL	12	по заданной маске

Функция

```
void pieslice(int x, int y, int a, int b, int R);
```

используется для построения секторов круга с заливкой. Координаты `x, y` — центр окружности, сектор рисуется от угла a до угла b.

Функция

```
void sector(int x, int y, int a, int b,
            int Rx, int Ry);
```

рисует сектор эллипса с заливкой. Координаты x, y — центр эллипса, a b — начальный и конечный углы сектора, Rx, Ry — радиусы эллипса в горизонтальном и вертикальном направлениях.

9.2.5. Вывод текста в графическом режиме

Для вывода на экран текстовых надписей в графическом режиме используются функции `outtext()` и `outtextxy()`. Функция

```
void outtext(char *string);
```

выводит строку текста, начиная с текущего положения указателя. Например,

```
outtext("Для продолжения нажмите Enter");  //.
```

Если необходимо указать точку начала вывода, целесообразно использовать функцию

```
void outtextxy(int x, int y, char *string);
```

где x, y — координаты точки начала вывода текста. Например,

```
outtextxy(50,100,"Для продолжения нажмите Enter");//.
```

Следует заметить, что для вывода на экран численных данных их необходимо предварительно перевести в строковый вид. Это можно сделать с помощью функции

```
sprintf (char *S, "Форматная строка", ...);
```

которая аналогична функции ptintf(), только вывод производится в строку S, указанную первым аргументом. Например,

```
x=3.14159;
char *S;
sprintf(S, "%f",x)
outtextxy(20, 40,S);  //.
```

Функция

```
void settextstyle(int font, int B, int size);
```

предназначена для управления параметрами шрифта при выводе текста в графическом режиме. Здесь `font` — номер шрифта (по умолчанию DEFAULT_FONT=0), `b` — направление (горизонтальное `b=0` или вертикальное `b=1`), `size` — размер выводимых символов: при `size=1` — размер буквы 8x8 точек, при `size=2` размер буквы — 16x16 точек. Возможные шрифты, поддерживаемые библиотекой, и их номера перечислены в табл. 15.

Таблица 15

Номера шрифтов

Название	**Номер**
DEFAULT_FONT	0
TRIPLEX_FONT	1
SMALL_FONT	2
SANS_SERIF_FONT	3
GOTHIC_FONT	4
SCRIPT_FONT	5
SIMPLEX_FONT	6
TRIPLEX_SCR_FONT	7
COMPLEX_FONT	8
EUROPEAN_FONT	9
BOLD_FONT	10

Очистка графического окна и установка указателя в точку с координатами (0,0) осуществляет процедура

```
void cleardevice(void); //.
```

9.3. Построение графиков функций

В качестве примера напишем программу, которая рисует в графическом окне график функции $f(x) = x\sin(x)$.

Включим в программу заголовочные файлы библиотек

```
#include <graphics.h>
#include <math.h>  //.
```

Главная функция `main()` начинается инициализацией графического режима

```
int gd, gm;
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"");
```

назначается белый цвет фона и выполняется очиска экрана

```
setbkcolor(WHITE);
cleardevice();  //.
```

Будем различать экранную систему координат XOY с началом в левом верхнем углу окна и систему координат xoy, относительно которой будет построен график. Центр системы xoy находится в точке (X0,Y0) относительно системы координат XOY. Ось абсцисс ox направим вправо, ось ординат oy — вверх, а также выберем размеры единичных отрезков `edx`, `edy`. Тогда для любой точки с экранными координатами (X,Y) можно вычислить её координаты в системе xoy:

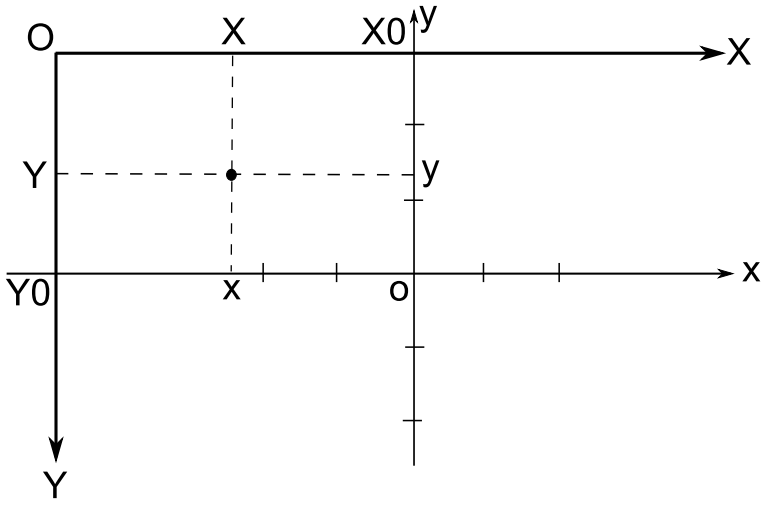

Рис. 22. Преобразование экранных координат

$x = (X - X0)/edx$, $y = (Y0 - Y)/edy$ (см. рис. 22). И наоборот, координаты точек графика (x,y) можно пересчитать в экранные координаты: `X=X0+edx*x, Y=Y0-edy*y`.

Далее необходимо нарисовать оси координат, относительно которых будет построен график. Для этого выбираем экранные координаты (X0,Y0) центра новой системы координат посередине экрана

```
int X0=getmaxx()/2;
int Y0=getmaxy()/2;
```

и рисуем оси

```
line(0,Y0,getmaxx(),Y0);
line(X0,0,X0,getmaxy());  //.
```

Перпендикулярно осям маленькими линиями изобразим засечки с шагом, равным единичным отрезкам

```
    int edx=20;
    int edy=15;
// текущий цвет - зелёный
    setcolor(LIGHTBLUE);
// переменная цикла
    int k;
// рисуем засечки
    // по оси x
    for(k=0;edx*k<X0;k++)
      {
       line(X0+k*edx,Y0-3,X0+k*edx,Y0+3);
       line(X0-k*edx,Y0-3,X0-k*edx,Y0+3);
      }
    // по оси y
    for(k=0;edy*k<Y0;k++)
      {
       line(X0-3,Y0+k*edy,X0+3,Y0+k*edy);
       line(X0-3,Y0-k*edy,X0+3,Y0-k*edy);
      }  //.
```

Далее приступаем к рисованию графика. Установливаем текущими розовый цвет и сплошной тип линии толщиной 3 точки

```
setcolor(LIGHTRED);
setlinestyle(SOLID_LINE,0,THICK_WIDTH);  //.
```

Объявляем переменные вещественного типа: `a,b` — границы области построения по оси *ox*, `x,y` — текущие координаты указателя, `delta` — шаг построения по оси *ox*

```
    float a,b,x,y,delta;
    // интервал дискретизации
    delta=0.1;
```

```
    // границы построения по оси x
    a=-15; b=15;  //.
```

Начальная точка построения графика имеет координаты (a,f(a)). Переместим в неё указатель (графический курсор).

```
// Вычисление начальных координат
    x=a;
    y=x*sin(x);
// перемещение указателя в начало графика
    moveto(X0+edx*x,Y0-edy*y);  //.
```

Для построения будем вычислять координаты следующей точки графика (x,f(x)) и рисовать линию между текущим положением указателя и вновь вычисленной точкой. Таким образом, график представляет собой ломаную линию, приближенную к реальному графику.

```
// цикл построения графика
    for(x=a;x<=b;x=x+delta)
       {
         y=x*sin(x);
         lineto(X0+edx*x,Y0-edy*y);
       };  //.
```

В завершение программы в графическое окно выводится заголовочная строка.

```
// установка стиля текста
    settextstyle(SANS_SERIF_FONT,0,2);
// вывод заголовка
    outtextxy(180,10,"График функции f(x)=x*sin(x)");
```

Ниже приведён полный текст программы построения графика, а результат её работы — графическое окно — показано на рис. 23.

```
#include <graphics.h>
#include <math.h>
int main()
{
// Инициализация графики
    int gd, gm;
    detectgraph(&gd,&gm);
    initgraph(&gd,&gm,"");
// цвет фона - белый
    setbkcolor(WHITE);
// очистка окна
    cleardevice();
// текущий цвет - черный
    setcolor(BLACK);
// координаты центра
    int X0=getmaxx()/2;
    int Y0=getmaxy()/2;
// оси координат
    line(0,Y0,getmaxx(),Y0);
    line(X0,0,X0,getmaxy());
// единичные отрезки в пикселах
    int edx=20;
    int edy=15;
// текущий цвет - зелёный
    setcolor(LIGHTBLUE);
// переменная цикла
    int k;
// рисуем засечки
    // по оси x
    for(k=0;edx*k<X0;k++)
      {line(X0+k*edx,Y0-3,X0+k*edx,Y0+3);
       line(X0-k*edx,Y0-3,X0-k*edx,Y0+3);}
```

```
    // по оси y
    for(k=0;edy*k<Y0;k++)
      {line(X0-3,Y0+k*edy,X0+3,Y0+k*edy);
       line(X0-3,Y0-k*edy,X0+3,Y0-k*edy);}
// текущий цвет - розовый
    setcolor(LIGHTRED);
// стиль линии - сплошная толстая
    setlinestyle(SOLID_LINE,0,THICK_WIDTH);
// описание переменных
    float a,b,x,y,delta;
    // интервал дискретизации
    delta=0.1;
    // границы построения по оси x
    a=-15; b=15;
// Вычисление начальных координат
    x=a;
    y=x*sin(x);
// перемещение указателя в начало графика
    moveto(X0+edx*x,Y0-edy*y);
// цикл построения графика
    for(x=a;x<=b;x=x+delta)
       {y=x*sin(x);
         lineto(X0+edx*x,Y0-edy*y);};
// установка стиля текста
    settextstyle(SANS_SERIF_FONT,0,2);
// вывод заголовка
  outtextxy(180,10,"График функции f(x)=x*sin(x)");
  getch();  // ожидание
// закрытие графического окна
    closegraph();
    return 0;
}  //.
```

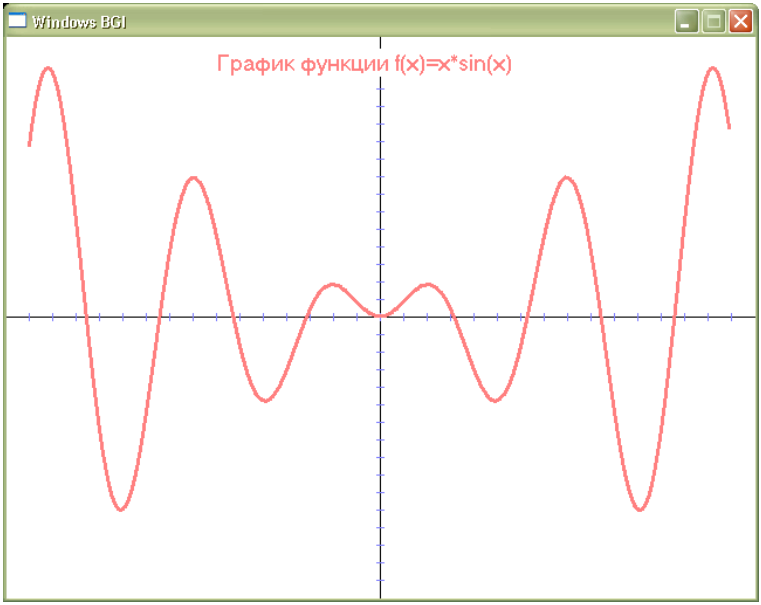

Рис. 23. Графическое окно

9.4. Практическое задание

Дана n+1 пара значений (x, y), связанных функциональной зависимостью $f(x)$ вашего варианта. Значения x выбраны в n+1 равноотстоящих точках отрезка $[a, b]$. Необходимо построить интерполяционный многочлен $P_n(x)$. Для этого:

1. Вычислите значения функции $f(x)$ в равноотстоящих точках отрезка $[a, b]$. Тем самым будут заполнены два массива x и y, которые можно интерпретировать как экспериментально измеренные значения каких-нибудь физических величин.
2. Вычислите разделённые разности по значениям функ-

ции в $n+1$ узле.

3. Запрограммируйте вычисление интерполяционного многочлена $P_n(x)$ по схеме Горнера.
4. Постройте в графическом окне графики исходной функции $f(x)$ и интерполяционного многочлена $P_n(x)$ на отрезке $[a-\eta, b+\eta]$. Значение $\eta > 0$ выбирите таким, чтобы область построения графика по оси x была в $1.5 \div 2$ раза больше отрезка, на котором выбиралить узлы дискретизации.
5. Исследуйте поведение интерполяционного многчлена при различном количестве узлов дискретизации (4,5,7,10 точек) .
6. Смоделируйте погрешность задания значений функции в узлах дискретизации путём добавления к значениям функции случайных чисел.

Таблица 16

Варианты заданий

№	$f(x)$	$[a,b]$	№	$f(x)$	$[a,b]$
1	e^x	$[-1,1]$	9	$1/(1-x^2)$	$[2,5]$
2	e^{-x}	$[-2,2]$	10	$x^2\ln(x)$	$[1,5]$
3	$\sin(x)$	$[-\pi/2,\pi/2]$	11	$\ln(\lvert x\rvert)$	$[-2,2]$
4	$\cos(x)$	$[0,4\pi]$	12	$\lvert\sin(x)\rvert$	$[-\pi/2,\pi/2]$
5	$\mathrm{tg}(x)$	$[-\pi/4,\pi/4]$	13	$\ln(x^2+1)$	$[-2,2]$
6	$1/x$	$[1,3]$	14	$e^{-\lvert x\rvert}$	$[-1,1]$
7	$1/x^2$	$[1,3]$	15	$\sqrt{x^3}$	$[0,3]$
8	$\sqrt{x}$	$[0,5]$	15	$\cos(x)-\sin(x)$	$[-4,4]$

10. СОЗДАНИЕ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ СРЕДСТВАМИ БИБЛИОТЕКИ Qt

Подавляющее большинство современные программных продуктов обладают так называемым графическим интерфейсом пользователя (Graphics User Interface, GUI). В свою очередь современные операционные системы предоставляют возможности создания GUI. Так, например, Windows имеет в своём составе библиотеку API (Application Programming Interface) функций. Однако, их использование требует большого практического опыта и глубоких знаний особенностей работы операционной системы. Для облегчения написания программ созданы другие библиотеки более доступных инструментов: MCF (Microsoft Foundation Classes), VCL (Visual Component Library) фирма Borland и др.

Библиотека Qt представляет собой набор инструментальных средств для построения графического интерфейса пользователя и обладает как минимум двумя преимуществами по сравнению с другими библиотеками. Во-первых она является кроссплатформенной. Это означает, что программы на базе Qt выглядят одинаково на разных операционных системах. Во-вторых существует версия Qt, распространяемая свободно. В данной лабораторной работе будут описаны лишь некоторые возможности библиотеки Qt, поскольку для полного её описания потребуется не одна такая книга. Желающим более глубоко изучить Qt можно посоветовать книги [17, 18, 19].

Библиотека Qt построена в виде библиотеки классов. Поэтому перед её изучением целесообразно также подробно ознакомиться с принципами объектно ориентированного программирования [6, 7, 8, 9].

10.1. Создание главного окна приложения

Программы с графическим интерфейсом пользователя состоят как минимум из одного главного окна, которое появляется при запуске программы. В ходе работы возможно открытие и закрытие других окон. При закрытии главного окна завершается работа всей программы. Основной целью использования графического интерфейса является взаимодействие с пользователем. Поэтому программа должна выполнять так называемый «цикл обработки событий». События происходят от действий пользователя: перемещений и щелчков мышью, нажатий клавиш на клавиатуре и т.д. Информация о происшедшем событии передаётся программе, которая может содержать подпрограмму обработки этого события.

Для реализации перечисленных свойств в библиотеке Qt реализован класс QApplication. Подавляющее большинство программ на Qt в самом начале создают экземпляр этого класса и используют его метод `exec()` для запуска цикла обработки событий. Поэтому обычно используется следующая структура Qt-программы:

```
#include <QApplication>
...
int main(int argc, char* argv[])
{
    QApplication app(argc, argv);
    ...
    return app.exec();
}  //.
```

Директива `#include <QApplication>` включает в текст программы заголовочный файл `QApplication.h`. При использо-

вании ещё каких-либо классов необходимо включать в программу заголовочные файлы каждого из них или воспользоваться более общим файлом QtGui.h. Аргументы функции `main` служат для взаимодействия программы с операционной системой. Операционная система позволяет запускать программу различными способами, например, из командной строки или с помощью ярлыка. При этом после имени программы могут быть указаны агрументы, количество которых заранее неизвестно. Все аргументы передаются программе в виде массива строк `argv`, а их количество — в виде целой переменной `argc`. Дальнейшая судьба этих переменных зависит от самой программы. Строка `QApplication app(argc, argv);` создаёт экземпляр `app` класса `QApplication`. В качестве аргументов конструктору класса передаётся массив строк argv и размер этого массива argc. Это обстоятельсто даёт возможность управлять внешним видом приложения. При запуске Qt-программы может быть указан параметр `-style=ИмяСтиля`. В качестве имени стиля допускается использовать Windows, CDE, Motif, Plastique или Cleanlooks в любой операционной системе. Функция `exec()` представляет собой метод класса QApplication и реализует цикл обработки событий. Она возвращает целый результат — код завершения программы, передаваемый операционной системе. При нормальном завершении программы код равен нулю.

10.1.1. Компиляция Qt программы

Компиляция Qt программы обладает некоторыми особенностями. Как правило программа состоит из нескольких файлов, расположенных в одном каталóге, который будем называть каталóгом проекта. Достаточно просто выполнять компиляцию с помощью командной строки. Для этого необ-

ходимо:

1. В командной строке сделать текущим каталóг проекта. Библиотека Qt при установке создаёт ярлык «Qt Command Prompt» для запуска командной строки с уже настроенными параметрами компиляции. Можно также пользоваться командной строкой оболочки Far manager или консолью Linux.
2. Выполнить команду `qmake -project`. При этом будет создан файл проекта с расширением `pro`.
3. Выполнить команду `qmake`. В результате будут создан Makefile.
4. Выполнить команду `make` для компиляции.

Описанный способ компиляции является самым универсальным. Можно также компилировать программы с помощью интегрированных сред. Самой удобной является IDE Kdevelop операционной системы Linux. Интегрированную среду Code::Blocks можно использовать только совместно с плагином QtWorkBench. Но усилия, потраченные на установку и настройку плагина сравнимы с компиляцией с помощью командной строки. Поэтому вопрос использования Qt совместно с интегрированными средами программирования не будем рассматривать в данной книге.

10.1.2. Виджеты

Визуальные элементы управления приложением, (кнопки, строки ввода, текстовые надписи и т.д.) называются виджетами. Каждому виджету соответствует в библиотеке Qt свой класс. Например, для отображения текстовых надписей используется класс QLabel (метка). Каждый виджет по сути представляет собой окно, поэтому главное окно приложения можно строить на основе какого-нибудь виджета. Создадим

в программе метку — экземпляр класса QLabel:

```
QLabel Lab("Hello!!!");
```

изменим её размер с помощью метода `resize` и покажем метку методом `show`

```
Lab.resize(50,30);
Lab.show();  //.
```

В результате получаем следующий текст программы

```
#include <QtGui>
int main(int argc, char* argv[])
{
    QApplication app(argc, argv);
    QLabel Lab("Hello!!!");
    Lab.resize(50,30);
    Lab.show();
    return app.exec();
}  //.
```

Аналогично можно создать приложение на основе других виджетов. В следующем примере программа создаётся на основе виджета класса QPushButton, который представляет собой кнопку.

```
#include <QtGui>
int main(int argc, char* argv[])
{
    QApplication app(argc, argv);
    QPushButton *Btn = new QPushButton("Button");
    Btn->resize(50,30);
    Btn->show();
    return app.exec();
} //.
```

Здесь экземпляр класса `QPushButton` создаётся динамически с помощью операции `new`. В результате работа с вновь созданным объектом «кнопка» выполняется через указатель `Btn`. Классы всех виджетов являются потомками класса `QWidget`. Экземпляры этого класса можно использовать в качестве «подложки» главного окна приложения. Создадим окно на основе объекта класса `QWidget`

```
#include <QtGui>
int main(int argc, char* argv[])
{
    QApplication app(argc, argv);
    QWidget *wdt = new QWidget();
    wdt->resize(300,300);
    wdt->setWindowTitle("!!!Window!!!");
    wdt->show();
    return app.exec();
}  //.
```

В результате интерфейс программы представляет собой пустое окно размером 300 на 300 точек и заголовком «!!!Winwow!!!». На этом пустом окне можно размещать другие элементы управления. Разместим, например, две кнопки. Для этого динамически создадим объекты класса QPushButton

```
QPushButton *btn1 = new QPushButton("Button1");
QPushButton *btn2 = new QPushButton("Button2");
```

а виджет `wdt` назначим родителем созданных кнопок

```
btn1->setParent(wdt);
btn2->setParent(wdt);   //.
```

Далее переместим кнопки в указанные координаты относительно родительского виджета и покажем главное окно

```
btn1->move(10,10);
btn2->move(100,10);
wdt->show();  //.
```

Таким образом, получим следующий текст программы:

```
#include <QtGui>
int main(int argc, char* argv[])
{
    QApplication app(argc, argv);
    QWidget *wdt = new QWidget();
    wdt->resize(300,300);
    wdt->setWindowTitle("!!!Window!!!");
    QPushButton *btn1 = new QPushButton("Button1");
    QPushButton *btn2 = new QPushButton("Button2");
    btn1->setParent(wdt);
    btn2->setParent(wdt);
    btn1->move(10,10);
    btn2->move(100,10);
    wdt->show();
    return app.exec();
}  //.
```

Внешний вид главного окна показан на рис. 24. Однако, такое приложение не обладает никакой функциональностью, поскольку не предусмотрены действия, выполняемые при нажатии кнопок.

10.1.3. Автоматическое размещение виджетов

Ручное размещение виждетов внутри главного окна, применяемое в предыдущих примерах, обладает недостатком: при изменении размеров окна размещение виджетов

Рис. 24. Внешний вид окна с двумя кнопками

не подстраивается под новые размеры. Избавиться от этого недастатка позволяет применение так называемых менеджеров компоновки. Они представляют собой объекты-контейнеры, автоматически располагающие одни виджеты на поверхности другого виджета.

Библиотека Qt предоставляет три основных класса менеджеров компоновки: `QHBoxLayout` (для горизонтального размещения), `QVBoxLayout` (для вертикального размещения) и `QGridLayout` (для табличного размещения). Продемонстрируем их применение на примере.

Пусть необходимо разместить в главном окне две кнопки, две текстовые строки ввода и одну метку. Сначала отдельно создаём следующие не зависящие друг от друга объекты: приложение,

```
QApplication app(argc, argv);
```

виджет главного окна

```
QWidget *wdt = new QWidget();
```

две кнопки

```
QPushButton *btn1 = new QPushButton("Button1");
QPushButton *btn2 = new QPushButton("Button2");
```

две строки ввода

```
QLineEdit *edt1 = new QLineEdit("Text1");
QLineEdit *edt2 = new QLineEdit("Text2");
```

метка

```
QLabel *lb = new QLabel("Text Label");
```

два горизонтальных менеджера компоновки

```
QHBoxLayout *hlayout1 = new QHBoxLayout;
QHBoxLayout *hlayout2 = new QHBoxLayout;
```

один вертикальный менеджер компоновки

```
QVBoxLayout *vlayout = new QVBoxLayout;  //.
```

Затем назначаем размеры главного окна, его заголовок и способ выравнивания текста метки:

```
wdt->resize(300,150);
wdt->setWindowTitle("!!!Window!!!");
lb->setAlignment(Qt::AlignCenter);
```

И, наконец, вводим в действие менеджеры компоновки. Для этого строки ввода `edt1` и `edt2` добавляем с помощью метода `addWidget()` к первому горизонтальному менеджеру компоновки `hlayout1`, а кнопки `btn1` и `btn2` — ко второму `hlayout2`

```
hlayout1->addWidget(edt1);
hlayout1->addWidget(edt2);
hlayout2->addWidget(btn1);
hlayout2->addWidget(btn2);  //.
```

Теперь в менеджер вертикальной компоновки погружаем поочереди: `hlayout1` с двумя текстовыми строками, метку `lb` и `hlayout2` с двумя кнопками

```
vlayout->addLayout(hlayout1);
vlayout->addWidget(lb);
vlayout->addLayout(hlayout2);  //.
```

Здесь для добавления к одному межеджеру компоновки другого, использован метод `addLayout()`. Теперь объект `vlayout` необходимо разместить на главном окне `wdt` с помощью метода `setLayout()`

```
wdt->setLayout(vlayout);  //.
```

На рис. 25 изображено получившееся окно программы и штриховыми линиями выделены горизонтальные менеджеры компоновки и метка, сплошной линией показан вертикальный менеджер компоновки. Ниже приведён полный текст получившейся программы. Более подробно автоматическое размещение виджетов описано в книгах [17, 18, 19].

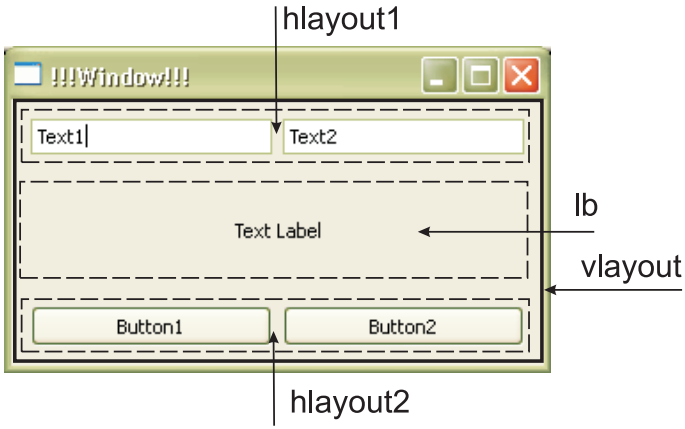

Рис. 25. Пример использования менеджеров компоновки

```
#include <QtGui>
int main(int argc, char* argv[])
{
    QApplication app(argc, argv);
    QWidget *wdt = new QWidget();
    QPushButton *btn1 = new QPushButton("Button1");
    QPushButton *btn2 = new QPushButton("Button2");
    QLineEdit *edt1 = new QLineEdit("Text1");
    QLineEdit *edt2 = new QLineEdit("Text2");
    QLabel *lb = new QLabel("Text Label");
    QHBoxLayout *hlayout1 = new QHBoxLayout;
    QHBoxLayout *hlayout2 = new QHBoxLayout;
    QVBoxLayout *vlayout = new QVBoxLayout;
    wdt->resize(300,150);
    wdt->setWindowTitle("!!!Window!!!");
    lb->setAlignment(Qt::AlignCenter);
    hlayout1->addWidget(edt1);
    hlayout1->addWidget(edt2);
```

```
    hlayout2->addWidget(btn1);
    hlayout2->addWidget(btn2);
    vlayout->addLayout(hlayout1);
    vlayout->addWidget(lb);
    vlayout->addLayout(hlayout2);
    wdt->setLayout(vlayout);
    wdt->show();
    return app.exec();
}  //.
```

10.1.4. Создание собственных виджетов

Окно программы, построенное в предыдущем пункте можно реализовать с помощью создания собственного виджета, который является наследником класса `QWidget`. Назовём класс нового виджета `TForm`. Объекты, расположенные на главном окне (в нашем случае две кнопки, две строки ввода и метка) будут считаться членами нового класса. Описание виджета `TForm` разместим в заголовочном файле `form.h`:

```
#include <QtGui>

#ifndef F_H_INCLUDED
#define F_H_INCLUDED

class TForm : public QWidget
{
public :
 TForm(); // конструктор

private :
  // виджеты главного окна
 QPushButton *btn1;
```

```
 QPushButton *btn2;
 QLineEdit *edt1;
 QLineEdit *edt2;
 QLabel *lb;
 QHBoxLayout *hlayout1;
 QHBoxLayout *hlayout2;
 QVBoxLayout *vlayout;
 };

#endif // F_H_INCLUDED.
```

Обсудим новые конструкции, встретившиеся в описании класса `TForm`. Директивы препроцессора `#ifndef`, `#define`, `#endif` служат здесь для предотвращения многократного включения заголовочного файла в текст программы.

Вообще директива *define* позволяет связать идентификатор с некоторой замещающей строкой (возможно пустой), например, строка

```
#define PI 3.14159
```

идентификатору PI ставит в соответствие значение 3,14159. Перед компиляций программы препроцессор выполнит замену в тексте программы замещаемых идентификаторов замещающими строками.

Директива *ifndef* управляет включением в программу фрагментов текста. Весь текст, расположенный между `#ifndef NAME` и `#endif` включается в программу при условии, что константа NAME *не* определена.

Класс `TForm` объявляется наследником класса `QWidget`. Общедоступным является конструктор класса `TForm()`. Закрытыми считаются объекты, расположенные на главном окне — кнопки, метки и т.д. Поскольку объекты предполага-

ется создавать динамически, членами класса являются указатели на них.

Реализацию методов класса разместим в файле с именем `form.cpp`:

```
#include "form.h"

TForm::TForm()
{
    btn1 = new QPushButton("Button1");
    btn2 = new QPushButton("Button2");
    edt1 = new QLineEdit("Text1");
    edt2 = new QLineEdit("Text2");
    lb = new QLabel("Text Label");
    hlayout1 = new QHBoxLayout;
    hlayout2 = new QHBoxLayout;
    vlayout = new QVBoxLayout;
    lb->setAlignment(Qt::AlignCenter);
    hlayout1->addWidget(edt1);
    hlayout1->addWidget(edt2);
    hlayout2->addWidget(btn1);
    hlayout2->addWidget(btn2);
    vlayout->addLayout(hlayout1);
    vlayout->addWidget(lb);
    vlayout->addLayout(hlayout2);
    this->setLayout(vlayout);
} //.
```

В описании класса присутствует только один метод — конструктор. В нём происходит создание объектов и размещение их на форме. Текст метода соответствует программе, разобранной в предыдущем пункте, и не нуждается в комментариях за исключением слова *this*. Под *this* понимается имя

объекта, которому принадлежит конструктор (или другой метод). Это связано с тем, что класс является лишь проектом объекта, его описанием. Самого объекта пока не существует, и имя его неизвестно.

Наконец, саму программу, использующую новый класс разместим в файле `main.cpp`:

```
#include <QtGui>
#include "form.h"

int main(int argc, char* argv[])
{
    QApplication app(argc, argv);
    TForm *wdt = new TForm();
    wdt->resize(300,150);
    wdt->setWindowTitle("!!!Window!!!");
    wdt->show();
    return app.exec();
}  //.
```

Здесь динамичнски создаётся объект `wdt` — экземпляр класса `TForm` — главное окно приложения. Поскольку создание объекта сопровождается выполнением его конструктора, все остальные виджеты будут созданые автоматически. На первый взгляд применение собственных виджетов приводит к необоснованному усложнению программы. Однако при наполнении программы функциональностью именно созданный нами виджет позволит добавлять к программе новые функции.

10.2. Наполнение программы функциональностью. Сигналы и слоты

Отличительной особенностью библиотеки Qt является наличие механизма сигналов и слотов, позволяющего связать объекты между собой. Считается, что объекты могут испускать сигналы. Одним из самых ярких примеров сигнала является телефонный звонок. Подняв трубку, человек выполняет «обработку» сигнала. С точки зрения программы сигнал представляет собой метод класса без реализации и не возвращающий никакого значения. Аргументы этого метода могут передать обработчику сигнала дополнительную информацию, как, например, телефон с определителем номера вместе с сигналом сообщает человеку дополнительную информацию — номер вызывающего абонента. Стандартные классы библиотеки Qt имеют все необходимые им сигналы, но при описании собственных классов можно вводить новые сигналы. Необходимым для этого условием является наличие ключевого слова `Q_OBJECT` в секции private описания класса (как правило в самом начале), например:

```
class TForm : public QWidget
{
Q_OBJECT
...
signals :
void NameSignal(void);
...
 };  //.
```

Здесь описан сигнал с именем `NameSignal` без аргументов. Причиной появления сигнала может быть какое-либо событие, например нажатие кнопки, перемещение ползунка, вы-

бор пункта меню и т.д. Можно также принудительно выслать сигнал с помощью ключевого слова

```
emit NameSignal();  //.
```

Поскольку сигналы могут высылаться только объектами того класса, которому они принадлежат, желательно предусмотреть метод, высылания сигнала

```
class TForm : public QWidget
{
Q_OBJECT
...
signals :
void NameSignal(void);
...
public :
void SendNameSignal(void)
   {
    emit NameSignal();
   }
...
 }; //.
```

Слоты — это обработчики сигналов. По сути слоты являются обычными методами класса, но ещё обладают свойством присоединяться к сигналам. Объекты библиотеки уже обладают необходимым набором слотов для реализации интерфейса программы. Но для наполнения программы нужной функциональностью приходится создавать собственные слоты.

Соединение сигналов со слотами происходит с помощью метода `connect()` класса `QObject` — самого верхнего в иерархии классов Qt

```
connect(Sender, SIGNAL(NameSignal()),
        Receiver, SLOT(NameSlot()));  //.
```

Здесь `Sender` — указатель на объект, который высылает сигнал, `NameSignal()` — высылаемый сигнал, `Receiver` — указатель на принимающий объект, `NameSlot()` — слот для обработки сигнала. Один сигнал может быть соединён с несколькими слотами.

В качестве примера адаптируем программу из предыдущего пункта к решению задачи варианта 14 лабораторной работы №1, в которой необходимо вычислить сумму квадратов N целых чисел $1^2 + 2^2 + 3^2 + 4^2 + ... + N^2$. Исходным данным является количество слагаемых N. Пусть величину N вводит пользователь в строку ввода `edt1`. При нажатии на кнопку `btn1` выполняется расчёт и результат выводится в строку ввода `edt2`. До расчёта метка `lb` отображает приглашение к вводу исходных данных, а после — сообщение о выполнении.

Для реализации перечисленных действий в классе TForm опишем слот `Btn1Click()`. В результате заголовочный файл `form.h` принимает следующий вид

```
#include <QtGui>
#ifndef F_H_INCLUDED
#define F_H_INCLUDED
class TForm : public QWidget
{
Q_OBJECT
public :
 TForm();

public slots :
void Button1Click(void);
```

```
private :
 QPushButton *btn1;
 QPushButton *btn2;
 QLineEdit *edt1;
 QLineEdit *edt2;
 QLabel *lb;
 QHBoxLayout *hlayout1;
 QHBoxLayout *hlayout2;
 QVBoxLayout *vlayout;
 };
#endif // F_H_INCLUDED.
```

Здесь добавлен макрос `Q_OBJECT` и прототип слота. В конструкторе класса выполним следующие изменения: сигнал `clicked()` кнопки `btn1` свяжем со слотом `Btn1Click()`, сигнал `clicked()` кнопки `btn2` свяжем со слотом `close()` главного окна, изменим текстовые надписи на объектах. Получим следующее содержание конструктора

```
TForm::TForm()
{
 btn1 = new QPushButton(
     QString::fromLocal8Bit("Посчитать"));
 btn2 = new QPushButton(
     QString::fromLocal8Bit("Выход"));
 edt1 = new QLineEdit("0");
 edt2 = new QLineEdit("0");
 lb = new QLabel(
     QString::fromLocal8Bit("Введите N"));
 hlayout1 = new QHBoxLayout;
 hlayout2 = new QHBoxLayout;
 vlayout = new QVBoxLayout;
```

```
 lb->setAlignment(Qt::AlignCenter);
 hlayout1->addWidget(edt1);
 hlayout1->addWidget(edt2);
 hlayout2->addWidget(btn1);
 hlayout2->addWidget(btn2);
 vlayout->addLayout(hlayout1);
 vlayout->addWidget(lb);
 vlayout->addLayout(hlayout2);
 this->setLayout(vlayout);
 // Соединение сигналов и слотов
 connect(btn1,SIGNAL(clicked()),
        this,SLOT(Button1Click()));
 connect(btn2,SIGNAL(clicked()),
        this,SLOT(close()));
} //.
```

Здесь для преобразования кодировки использована функция `QString::fromLocal8Bit()`. Это связано стем, что для в Qt приложениях для представления строк используется двухбайтовая кодировка Unicode, а при написании текста программы используется однобайтовая кодировка (в Windows — CP1251, в Linux — обычно koi8-r). Функция `fromLocal8Bit()` является методом класса `QString` и предназначена для преобразования строк с однобайтной системной кодировкой в формат строк класса `QString`.

Реализацию слота разберём несколько подробнее. Сначала необходимо получить исходные данные — значение переменной N. Вводимые пользователем данные представляют собой текстовую строку. Для её считывания объект `edt1` имеет метод `text()`, который возвращает объект класса `QString`. Класс `QString` обладает методами преобразования строки в число: `toInt()`, `toDouble()`. Поэтому для считвания исходных данных целесообразно использовать присваи-

вание

```
N=edt1->text().toInt();
```

где выражение `edt1->text()` получает объект класса `QString`, а `.toInt()` вызывает его метод преобразования текста в целое число. Расчет суммы квадратов целых чисел не вызывает затруднений и может быть выполнен оператором

```
for(k=1;k<=N;k++) {s=s+k*k;};  //.
```

Результат вычислений содержит переменная s. Для вывода в строку ввода `edt2` её необходимо преобразовать в строковый вид. Создадим вспомогательный объект `tmp` класса `QString` и с помощью метода `setNum()` сохраним текстовое представление целой переменной s в объекте `tmp`. И, наконец, с помощью метода `setText()` объекта `edt2` выводим текст из объекта `tmp` и меняем надпись метки `lb`

```
QString tmp;
 tmp.setNum(s);
 edt2->setText(tmp);
 lb->setText(
     QString::fromLocal8Bit("Выполнено"));
```

В результате получим следующую реализацию слота

```
void TForm::Button1Click(void)
{
 int N;
 N=edt1->text().toInt();
 int k;
 int s=0;
 for(k=1;k<=N;k++) {s=s+k*k;};
```

```
 QString tmp;
 tmp.setNum(s);
 edt2->setText(tmp);
 lb->setText(
     QString::fromLocal8Bit("Выполнено"));
} //.
```

Аналогично можно создавать слоты, выполняемые при нажатии других кнопок.

10.3. Создание интерфейса с помощью QtDesigner

Библиотека Qt комплектуется программой Qt Designer, предназначенной для конструирования пользовательского интерфейса. Рассмотрим кратко методику построения программы с применением Qt Designer. При запуске программы предлагается создать новое окно на основе одного из шаблонов: диалоговое окно с кнопками внизу, вверху, без кнопок, главное окно или пустой виджет. Чтобы оценить эффективность использования Qt Designer создадим с его помощью интерфейс программы, рассмотренной в предыдущем пункте. Пусть окно программы будет основано на пустом виджете. Будем называть далее проектируемое окно формой. В левой части окна Qt Designer расположены панели с элементами управления, которые можно расположить на форме. В правой части окна Qt Designer расположены панели инспеткора объектов, редактора свойств и т.д. Следует обратить внимание на имя формы Form, указанное в редакторе свойств. Оно потребуется в дальнейшем. Выполним далее следующую последовательность действий:

- разместим на форме две кнопки и назначим им с помощью редактора свойств имена btn1 и btn2;

- выделим их вместе и выберем в контекстном меню пункт Lay Out Horizontally;
- разместим на форме две строки ввода с именами edt1 и edt2;
- выделим их вместе и выберем в контекстном меню пункт Lay Out Horizontally;
- разместим на форме метку с именем lb;
- выделим все объекты и выберем в контекстном меню пункт Lay Out Vertically;
- сделаем правый щелчок на свободном месте формы и в контекстном меню выберем пункт Lay Out Vertically.

Сохраним проект под именем `forma.ui`. В результате в этом файле будет описан класс с именем `UiForm`. При компиляции средства библиотеки Qt преобразуют `forma.ui` в заголовочный файл `ui_forma.h` с описанием класса `UiForm` на языке C++. Чтобы воспользоваться окном, спроектированным в Qt Designer необходимо выполнить два действия. Во-первых, класс главного окна программы должет стать наследником не только класса `QWidget`, но и класса `UiForm`. Тогда заголовочный файл `form.h` станет значительно короче

```
#include <QtGui>
#include "ui_forma.h"

#ifndef F_H_INCLUDED
#define F_H_INCLUDED
class TForm : public QWidget, private Ui::Form
{
Q_OBJECT
public :
 TForm();
```

```
public slots :
void Button1Click(void);
};
#endif // F_H_INCLUDED.
```

Во-вторых, в конструкторе класса главного окна необходимо выполнить команду `setupUi(this)`. Тогда файл `form.cpp` имеет следующее содержимое

```
#include "form.h"

void TForm::Button1Click(void)
{
 int N;
 N=edt1->text().toInt();
 int k;
 int s=0;
 for(k=1;k<=N;k++) {s=s+k*k;};
 QString tmp;
 tmp.setNum(s);
 edt2->setText(tmp);
 lb->setText(QString::fromLocal8Bit("Выполнено"));
}

TForm::TForm()
{
 setupUi(this);
 connect(btn1,SIGNAL(clicked()),
         this,SLOT(Button1Click()));
 connect(btn2,SIGNAL(clicked()),
         this,SLOT(close()));
}
```

Как видно из листингов, использование средств Qt Designer позволяет несколько сократить текст программы.

В данном разделе рассмотрена лишь малая часть возможностей, которые предоставляет библиотека Qt. Освоение этой малой части должно быть достаточным для выполнения следующих двух лабораторных работ. Желающим более полно освоить программирование с использованием Qt можно посоветовать изучение книг [17, 18, 19].

11. ЛАБОРАТОРНАЯ РАБОТА №6 ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ

Цель работы:

- ознакомление с методами численного интегрирования;
- приобретение практических навыков построения пользовательского интерфейса средствами библиотеки Qt.

11.1. Основные методы численного интегрирования

В практической работе радиоинженера зачастую возникает необходимость вычисления интегралов, которые либо трудно, либо невозможно вычислить аналитически.

Определённый интеграл

$$I = \int_a^b f(x)dx \tag{5}$$

можно трактовать как площадь фигуры под графиком подынтегральной функции, то есть как площадь криволинейной трапеции. Многие методы численного интегрирования основаны на разбиении интервала интегрирования на более мелкие отрезки, на которых можно применить простые формулы для площади маленькой трапеции. Затем площади маленьких трапеций складываются.

Разобьем, например, отрезок интегрирования на N равных отрезков дискретизации длиной $h = (b - a)/N$ точками $a = x_0, x_1, ..., x_N = b$. Если отрезки дискретизации довольно малы, то на каждом отрезке площадь трапеции можно приближённо заменить площадью прямоугольника. Прямоугольники можно строить, опираясь на левую или правую

сторону отрезка (см. рис. 26). Отсюда получаем метод левых или правых прямоугольников. Ясно, что с увеличением числа отрезков, сумма площадей прямоугольников все с большей точностью будет совпадать с площадью криволинейной трапеции.

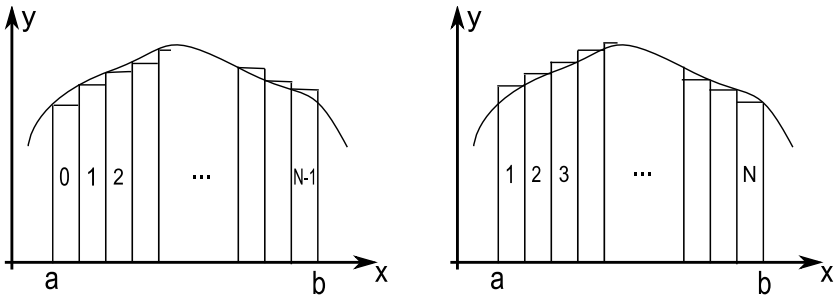

Рис. 26. Иллюстрация методов прямоугольников

Площадь маленького прямоугольника зависит от его номера k и равна $S_k = hf(x_k) = hf(a + hk)$. Тогда для приближённого вычисления интеграла справедливы формулы левых прямоугольников

$$I \approx \sum_{k=0}^{N-1} S_k = \frac{b-a}{N} \sum_{k=0}^{N-1} f(a + hk) \tag{6}$$

и правых прямоугольников

$$I \approx \sum_{k=1}^{N} S_k = \frac{b-a}{N} \sum_{k=1}^{N} f(a + hk). \tag{7}$$

Если на каждом отрезке площадь криволинейной трапеции приближённо заменить площадью прямолинейной трапеции (см рис. 27), то можно вычислить интеграл точнее.

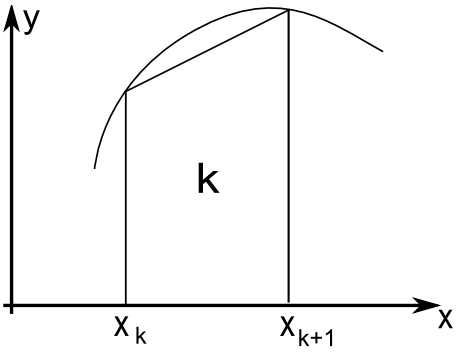

Рис. 27. Иллюстрация метода трапеций

Площадь прямоугольной трапеции с номером k равна

$$S_k = h[f(x_k) + f(x_{k+1})]/2 = h[f(a + hk) + f(a + h(k + 1))]/2,$$

а формула трапеций для вычисления интеграла имеет вид

$$I \approx \sum_{k=0}^{N-1} S_k = \frac{b-a}{2N} \sum_{k=0}^{N-1} [f(x_k) + f(x_{k+1})] =$$

$$= \frac{b-a}{N} \left[\sum_{k=1}^{N-1} f(a + kh) + \frac{f(a) + f(b)}{2} \right]. \qquad (8)$$

Выражения приближённого вычисления определённых интегралов (6) – (8) называют квадратурными формулами. Математически методика их получения как правило сводится к разбиению интервала интегрирования на много мелких подынтервалов и представления интеграла (5) в виде

$$I = \int_a^b f(x)dx = \sum_{k=0}^{N-1} \int_{x_i}^{x_{i+1}} f(x)dx. \qquad (9)$$

Далее подынтегральная функция на каждом подынтервале аппроксимируется некоторой другой функцией $g_k(x)$, интеграл от которой легко вычисляется аналитически. Если аппроксимировать функцию $f(x)$ константой $g_k(x) = f(x_k)$ или $g_k(x) = f(x_{k+1})$, то получаем формулы прямоугольников (6) и (7). Если аппроксимировать функцию прямой, проведённой через точки $(x_k, f(x_k))$ и $(x_{k+1}, f(x_{k+1}))$, то получаем формулу трапеций (8). В общем виде квадратурную формулу можно записать в виде

$$I \approx \sum_{k=0}^{N} A_k f(x_k), \tag{10}$$

где коэффициенты A_k определяются способом аппроксимации подынтегральной функции. Перечислим некоторые квадратурные формулы, основанные на (10).

Формула Симпсона (формула парабол)

$$I \approx \frac{b-a}{3N}\left[4\sum_{k=1}^{N/2} f(x_{2k-1}) + 2\sum_{k=1}^{N/2-1} f(x_{2k}) + f(a) + f(b)\right]. \tag{11}$$

Значение N предполагается чётным.

Формула Гуасса

$$I \approx \frac{b-a}{2}\sum_{k=1}^{N} H_k f(x_k), \quad x_k = \frac{a+b}{2} + \frac{b-a}{2}t_k \tag{12}$$

основана на выборе неравноотстоящих узлов дискретизации t_k, совпадающих с нулями полиномов Лежандра порядка N на стандартном интервале $[-1, 1]$. Значения узлов t_k и весов H_k для разных N приведены в табл. 17.

Таблица 17

Узлы и веса квадратурной формулы Гаусса

$N = 1$	$t_1 = 0.5$	$H_1 = 2$
$N = 2$	$-t_1 = t_2 = 0.577350269$	$H_1 = H_2 = 2$
$N = 3$	$-t_1 = t_3 = 0.774596669$	$H_1 = H_3 = 0.555555555$
	$t_2 = 0$	$H_2 = 0.888888888$
$N = 4$	$-t_1 = t_4 = 0.861136311$	$H_1 = H_4 = 0.347854845$
	$-t_3 = t_2 = 0.339981043$	$H_2 = H_3 = 0.652145155$
$N = 5$	$-t_1 = t_5 = 0.906179846$	$H_1 = H_5 = 0.236926885$
	$-t_2 = t_4 = 0.538469310$	$H_2 = H_4 = 0.478628670$
	$t_3 = 0$	$H_3 = 0.568888888$

11.2. Практическое задание

Вычислите численно всеми рассмотренными методами интеграл, заданный в Вашем варианте. Сравните полученный результат с найденным аналитически. Предусмотрите графический интерфейс пользователя.

11.2.1. Варианты заданий

1. $\int\limits_{-2}^{2}(1-2|x|)dx.$
2. $\int\limits_{-2}^{2}|\cos(2x)|dx.$
3. $\int\limits_{0}^{5}\sqrt{x}dx.$
4. $\int\limits_{1}^{4}\ln(x)dx.$
5. $\int\limits_{-3}^{3}\exp(-x^2)dx.$
6. $\int\limits_{0}^{\pi}\cos^2(x)dx.$
7. $\int\limits_{0}^{4}\ln(x^2+1)dx.$
8. $\int\limits_{1}^{8}(x^2+x)dx.$
9. $\int\limits_{1}^{5}\ln(x)dx.$
10. $\int\limits_{-2}^{2}(x^3+x^2+1)dx.$

11. $\int_{-4}^{1} e^x dx.$

12. $\int_{0}^{10} x \exp(-x^2) dx.$

13. $\int_{0}^{\pi} \sin(x) dx.$

14. $\int_{-10}^{0} x^3 \exp(-x^2) dx.$

15. $\int_{-\pi/2}^{\pi/2} \cos(x) dx.$

16. $\int_{-5}^{5} (x^2 + |x|) dx.$

12. ЛАБОРАТОРНАЯ РАБОТА №7 ПРИБЛИЖЁННОЕ РЕШЕНИЕ НЕЛИНЕЙНЫХ УРАВНЕНИЙ

Цель работы:

- ознакомление с методами численного решения нелинейных уравнений;
- закрепление практических навыков построения пользовательского интерфейса средствами библиотеки Qt.

12.1. Основные методы численного решения уравнений

В практической работе радиоинженера зачастую возникает необходимость решения нелинейных уравнений, которые либо трудно, либо невозможно решить аналитически. Решение таких уравнений можно возложить на ЭВМ. Поэтому необходимо иметь представление о методах решения уравнений на ЭВМ. Любое уравнение можно представить в виде

$$f(x) = 0,$$

где $f(x)$ — некоторая нелинейная функция. Так, например, для уравнения $x^3 - 5x^2 + 2x = -5$ имеем

$$f(x) = x^3 - 5x^2 + 2x + 5.$$

Значение x при котором справедливо равенство $f(x) = 0$ называется корнем уравнения. Графически корни уравнения представляют собой абсциссы точек пересечения графика функции $f(x)$ с осью OX, то есть корни уравнения — это нули функции $f(x)$.

Процесс отыскания корней делиться на два этапа:
1. Отделение корней, т.е. определение отрезка содержащего

один корень.
2. Уточнение корня с заданной точностью.

Для *первого этапа* нет общих методов решения, отрезки определяются или табуляцией, или исходя из физического смысла, или аналитическими методами. Иногда выявить отрезки, на которых находятся корни, можно предварительным исследованием функции $f(x)$. Признаком того, что на отрезке $[a, b]$ находится корень уравнения является неравенство $f(a)f(b) < 0$. В примере кубического уравнения функция $f(x) = x^3 - 5x^2 + 2x + 5$ представляет собой кубическую параболу, которая может иметь один максимум и один минимум. Для того, чтобы найти положения максимума и минимума приравняем нулю производную функции $f(x)$:

$$3x^2 - 10x + 2 = 0.$$

Решим полученное уравнение: $x_1 \approx 3.12$, $x_2 \approx 0.21$. Поскольку $f(0.21) > 0$, $f(3.12) < 0$, $f(a)f(b) < 0$, то на отрезке $[0.21; 3.12]$ находится хотя бы один корень. Из общего вида функции $f(x)$ ясно, что найденный отрезок содержит только один корень. Для двух других корней отрезки придётся подобрать. Воспользуемся тем, что при $x \in (-\infty; 0.21)$ и $x \in (3.12; +\infty)$ функция $f(x)$ возрастает. Следовательно, если взять достаточно маленький x, например, $x = -2$, то $f(-2) < 0$ и отрезок $[-2; 0.21]$ содержит корень. Аналогично подберём довольно большой $x = 5$, так что $f(5) > 0$. Тогда отрезок $[3.12; 5]$ также содержит корень.

При других функциях $f(x)$ подобное рассуждение, возможно, провести не удастся. Тогда можно выбрать произвольные достаточно маленькие a и b, и постепенно смещать их вправо, пока не будет найден нужный отрезок. Критерием пригодности отрезка $[a, b]$ является неравенство $f(a)f(b) < 0$.

Второй этап — уточнение корня выполняется различ-

ными итерационными методами, суть которых в том, что строится числовая последовательность x_k сходящихся к корню. Выходом из итерационного процесса является одно из двух условий: $|f(x_k)| < \varepsilon$ или $|x_k - x_{k-1}| < \varepsilon$. Таким образом после нахождения отрезков локализации корней необходимо сужать эти отрезки до тех пор, пока либо их длина не станет меньше заранее заданной величины, которую будем называть точностью, например $\varepsilon = 10^{-6}$, либо значение функции $f(x)$ не приблизится к нулю с точностью ε. Рассмотрим наиболее употребляемые на практике методы: дихотомии (деления отрезка пополам), итераций и касательных.

12.1.1. Метод половинного деления

Дана монотонная, непрерывная функция $f(x)$, которая содержит корень на отрезке $[a, b]$, где $b > a$. Определить корень с точностью ε, если известно, что $f(a)f(b) < 0$.

Процесс уменьшения длины отрезка можно произвести следующим образом:
1) находим середину отрезка $d = (a + b)/2$,
2) находим значения функции $f(a)$, $f(b)$, $f(d)$,
3) если $f(a)f(d) < 0$, то корень находится в левой половине отрезка $[a, b]$. Переносим правую границу отрезка в его середину $b = d$; иначе переносим левую границу отрезка в его середину $a = d$ (см. рис. 28).
4) Если длина $|b - a| > \varepsilon$, то переходим к пункту 1.

Как только длина отрезка становится меньше необходимой точности $|b-a| < \varepsilon$, можем корнем уравнения считать любую точку отрезка, например, его середину.

12.1.2. Метод хорд

Процесс уменьшения длины отрезка происходит быстрее, если делить его не пополам, а используя хорду. Для

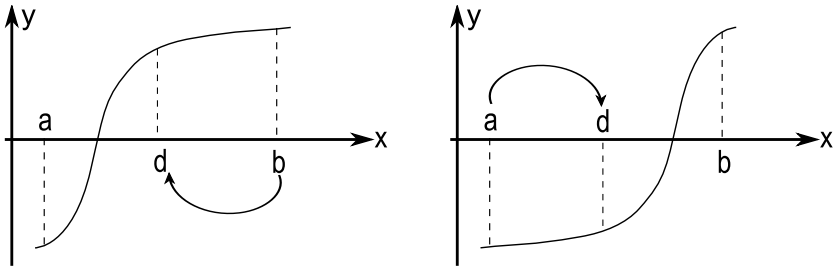

Рис. 28. Иллюстрация метода половинного деления

этого:
1) мысленно проводим прямую между точками $(a, f(a))$ и $(b, f(b))$.
Эта прямая пересекает ось X в точке $d = a-(b-a)f(a)/[f(b)-f(a)]$. Этой точкой отрезок $[a, b]$ разделяется на 2 части,
2) находим значения функции $f(a)$, $f(b)$, $f(d)$,
3) если $f(a)f(d) < 0$, то корень находится в левой части отрезка $[a, b]$. Переносим правую границу отрезка в точку d $(b = d)$; иначе переносим левую границу отрезка в точку d $(a = d)$.
4) Если длина $|b - a| > \varepsilon$, то переходим к пункту 1.

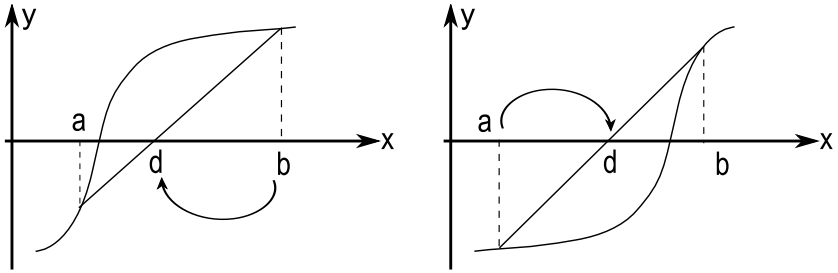

Рис. 29. Иллюстрация метода хорд

При достижении длины отрезка меньше необходимой

точности $|b-a| < \varepsilon$, выбираем в качестве корня уравнения любую точку отрезка, например, его середину.

12.1.3. Метод итераций

Заменим уравнение $f(x) = 0$ равносильным уравнением

$$x = \varphi(x). \tag{13}$$

Графически корни уравнения (13) представляют собой абсциссы точек пересечения графиков функций $y = \varphi(x)$ и $y = x$. Выберем приближённое значение корня x_0 , принадлежащее отрезку $[a, b]$, подставим его в правую часть уравнения (13), получим новое значение

$$x_1 = \varphi(x_0). \tag{14}$$

Далее подставим x_1 в правую часть уравнения (13) получим:

$$x_2 = \varphi(x_1). \tag{15}$$

И так далее на шаге с номером n имеем

$$x_n = \varphi(x_{n-1}). \tag{16}$$

Если последовательность x_n является сходящейся при $n \to \infty$, то данный алгоритм позволяет определить искомый корень.

12.1.4. Метод касательных (Ньютона)

1) Выбираем грубое приближение корня x_0 (либо точку a, либо b).

2) Находим значение функции в точке x_0 и проводим касательную к графику $f(x)$. Точку пересечения касательной с осью абсцисс $x_1 = x_0 - f(x_0)/f'(x_0)$ считаем следующим приближением корня.

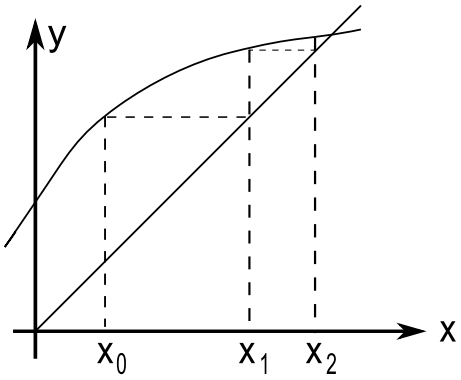

Рис. 30. Иллюстрация метода итераций

3) Определяем значение функции в точке x_1, через эту точку снова проводим касательную получим точку $x_2 = x_1 - f(x_1)/f'(x_1)$.
4) На шаге с номером n: $x_n = x_{n-1} - f(x_{n-1})/f'(x_{n-1})$.
Если последовательность x_n является сходящейся при $n \to \infty$, то данный алгоритм позволяет приближённо определить искомый корень.

В качестве примера приведём программу, которая находит корни уравнения $x^3 - 5x^2 + 2x = -5$ методом половинного деления.

```
#include <iostream.h>
#include <math.h>

// Функция, описывающая уравнение
float f(float x)
{return x*x*x-5*x*x+2*x+5;}

float eps=1e-6; // Точность
float a,b,d,res;
```

```
int main()
{
// Вводим с клавиатуры границы отрезка
// локализации корня
cout << "Введите границы отрезка: ";
cin >> a >> b;
if (f(a)*f(b)>0)
  {cout << "Отрезок задан неверно!";}
else
  {
    do // пока длина отрезка больше точности
    {
      d=(a+b)/2; // находим середину
      // выбираем половину отрезка с корнем
      if (f(a)*f(d)<0) {b=d;} else {a=d;}
    }
    while (fabs(b-a)>eps);
// находим результат
res=(b+a)/2;
cout << "Найдено рещение " << res << endl;
cout << "Проверка" << f(res) << endl;
  }
return 0;
} //.
```

Если уравнение настолько сложное, то найти отрезок локализации не удаётся, то можно предусмотреть в программе поиск отрезка, например, так:

```
a=-6;  b=a+0.1;
while (f(a)*f(b)>0)
{
```

```
  a=b;
  b=b+0.1;
}  //.
```

12.2. Практическое задание

Составьте программу, которая любыми двумя из рассмотренных методов находит все корни уравнения. Предусмотрите графический интерфейс пользователя.

12.2.1. Варианты заданий

1) $x^3 - x = 2x^2 - 2$
2) $x^3 + 12 = x(3x - 4)$
3) $x^3 = 4x(x - 1)$
4) $x^3 - x^2 - 8x = -12$
5) $-x^2 = \ln(x)$
6) $\ln(x) + 2 = x^3$
7) $x^2 = 2\sin(x) + 1$
8) $x^2 = 2\cos(2x) + 1$
9) $x^3 - 3x^2 - 5x = -12$
10) $x^3 = 2\sin(3x) + 1$
11) $\ln(x^2 + 1) = 1$
12) $\ln(x^4 + 1) = 2$
13) $\mathrm{arctg}(2x^2 - 2) = 1$
14) $2x^3 = 4x(x + 1)$
15) $x^4 + x = 2$
16) $x^4 + 5x = 3$
17) $x^4 + 5x = 3 - \ln(x)$
18) $x^2 + \sin(x) = 2 + x$
19) $x^2 + \sin(x) = 2 + x$
20) $x^2 + \sin(x) = \lg(x) + x$
21) $x^2 + \cos(x) = 7 + 4x$
22) $x^2 + \cos(x) = 7 - 4x$
23) $x^4 + \cos(x) = 5$
24) $3x^3 + 2x^2 = 5x + 2$
25) $3x^3 + 2x^2 = 3x + 1$
26) $x^3 - 10\sqrt{x} = -5$

ЗАКЛЮЧЕНИЕ

В учебном пособии изложены вопросы программирования на языках C и C++, изучаемые в курсе «Информатика» студентами радиотехнических специальностей вузов. Книга содержит краткое изложение теоретических сведений, адаптированное для выполнения лабораторных работ. Практические задания представлены семью лабораторными работами. Первые три из них посвящены общим вопросам программирования на C и C++. Описаны принципы работы с интегрированными средами, приведено краткое описание языка, методов работы с массивами, программирования подпрограмм. Рассмотрены способы хранения и обработки текстовых данных, взаимодействия с устройствами персонального компьютера. В результате выполнения первых трёх лабораторных работ студенты приобретают навыки программирования на C и C++.

Начиная с четвёртой лабораторной работы для построения пользовательского интерфейса проектируемой программы студенты изучают библиотеки PDCurses, WinBGIm и Qt. С их помощью строится пользовательский интерфейс и изучаются объектно-ориентированные возможности языка C++. Помимо внешней формы (графического пользовательского интерфейса), студенты осваивают некоторые темы численных методов: интерполяцию, численное интегрирование, приближённое решение уравнений.

Материал учебного пособия может быть использован при обучении программированию на языке C/C++, а практические задания к лабораторным работам — при изучении других языков программирования.

ПРИЛОЖЕНИЕ

Стандартная таблица символов ASCII (коды 0 – 31)

Код	Обозначение	Значение
0	nul	Нулевой символ
1	soh	Начало заголовка
2	stx	Начало текста
3	etx	Конец текста
4	eot	Конец передачи
5	enq	Запрос
6	ack	Подтверждение
7	bel	Звуковой сигнал
8	bs	Забой (шаг назад)
9	ht	Горизонтальная табуляция
10	lf	Перевод строки
11	vt	Вертикальная табуляция
12	ff	Новая страница
13	cr	Возврат каретки
14	so	Выключить сдвиг
15	si	Включить сдвиг
16	dle	Ключ связи данных
17	dc1	Управление устройством 1
18	dc2	Управление устройством 2
19	dc3	Управление устройством 3
20	dc4	Управление устройством 4
21	nak	Отрицательное подтверждение
22	syn	Синхронизация
23	etb	Конец передаваемого блока
24	can	Отказ
25	em	Конец среды
26	sub	Замена
27	esc	Ключ
28	fs	Разделитель файлов
29	gs	Разделитель группы
30	rs	Разделитель записей
31	us	Разделитель модулей

Стандартная таблица символов ASCII (коды 32 – 127)

Код		Символ	Код		Символ	Код		Символ	Код		Символ
дес.	шестн.		дес.	шестн.		дес.	шестн.		дес.	шестн.	
32	20		56	38	8	80	50	P	104	68	h
33	21	!	57	39	9	81	51	Q	105	69	i
34	22	"	58	3A	:	82	52	R	106	6A	j
35	23	#	59	3B	;	83	53	S	107	6B	k
36	24	$	60	3C	<	84	54	T	108	6C	l
37	25	%	61	3D	=	85	55	U	109	6D	m
38	26	&	62	3E	>	86	56	V	110	6E	n
39	27	'	63	3F	?	87	57	W	111	6F	o
40	28	(	64	40	@	88	58	X	112	70	p
41	29	)	65	41	A	89	59	Y	113	71	q
42	2A	*	66	42	B	90	5A	Z	114	72	r
43	2B	+	67	43	C	91	5B	[	115	73	s
44	2C	,	68	44	D	92	5C	\	116	74	t
45	2D	-	69	45	E	93	5D	]	117	75	u
46	2E	.	70	46	F	94	5E	^	118	76	v
47	2F	/	71	47	G	95	5F	_	119	77	w
48	30	0	72	48	H	96	60	`	120	78	x
49	31	1	73	49	I	97	61	a	121	79	y
50	32	2	74	4A	J	98	62	b	122	7A	z
51	33	3	75	4B	K	99	63	c	123	7B	{
52	34	4	76	4C	L	100	64	d	124	7C	|
53	35	5	77	4D	M	101	65	e	125	7D	}
54	36	6	78	4E	N	102	66	f	126	7E	~
55	37	7	79	4F	O	103	67	g	127	7F	

Кодовая таблица CP866 (коды 128 – 255)

Код		Символ	Код		Символ	Код		Символ	Код		Символ
дес.	шестн.		дес.	шестн.		дес.	шестн.		дес.	шестн.	
128	80	**А**	160	A0	**а**	192	C0	└	224	E0	**р**
129	81	**Б**	161	A1	**б**	193	C1	┴	225	E1	**с**
130	82	**В**	162	A2	**в**	194	C2	┬	226	E2	**т**
131	83	**Г**	163	A3	**г**	195	C3	├	227	E3	**у**
132	84	**Д**	164	A4	**д**	196	C4	─	228	E4	**ф**
133	85	**Е**	165	A5	**е**	197	C5	┼	229	E5	**х**
134	86	**Ж**	166	A6	**ж**	198	C6	╞	230	E6	**ц**
135	87	**З**	167	A7	**з**	199	C7	╟	231	E7	**ч**
136	88	**И**	168	A8	**и**	200	C8	╚	232	E8	**ш**
137	89	**Й**	169	A9	**й**	201	C9	╔	233	E9	**щ**
138	8A	**К**	170	AA	**к**	202	CA	╩	234	EA	**ъ**
139	8B	**Л**	171	AB	**л**	203	CB	╦	235	EB	**ы**
140	8C	**М**	172	AC	**м**	204	CC	╠	236	EC	**ь**
141	8D	**Н**	173	AD	**н**	205	CD	═	237	ED	**э**
142	8E	**О**	174	AE	**о**	206	CE	╬	238	EE	**ю**
143	8F	**П**	175	AF	**п**	207	CF	╧	239	EF	**я**
144	90	**Р**	176	B0	░	208	D0	╨	240	F0	**Ё**
145	91	**С**	177	B1	▒	209	D1	╤	241	F1	**ё**
146	92	**Т**	178	B2	▓	210	D2	╥	242	F2	Є
147	93	**У**	179	B3	│	211	D3	╙	243	F3	є
148	94	**Ф**	180	B4	┤	212	D4	╘	244	F4	
149	95	**Х**	181	B5	╡	213	D5	╒	245	F5	ї
150	96	**Ц**	182	B6	╢	214	D6	╓	246	F6	Ў
151	97	**Ч**	183	B7	╖	215	D7	╫	247	F7	ў
152	98	**Ш**	184	B8	╕	216	D8	╪	248	F8	°
153	99	**Щ**	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	**Ъ**	186	BA	║	218	DA	┌	250	FA	·
155	9B	**Ы**	187	BB	╗	219	DB	█	251	FB	√
156	9C	**Ь**	188	BC	╝	220	DC	▄	252	FC	№
157	9D	**Э**	189	BD	╜	221	DD	▌	253	FD	¤
158	9E	**Ю**	190	BE	╛	222	DE	▐	254	FE	■
159	9F	**Я**	191	BF	┐	223	DF	▀	255	FF	

Кодовая таблица CP1251 (коды 128 – 255)

Код		Символ	Код		Символ	Код		Символ	Код		Символ
дес.	шестн.		дес.	шестн.		дес.	шестн.		дес.	шестн.	
128	80	**Ђ**	160	A0		192	C0	**А**	224	E0	**а**
129	81	**Ѓ**	161	A1	**Ў**	193	C1	**Б**	225	E1	**б**
130	82	**‚**	162	A2	**ў**	194	C2	**В**	226	E2	**в**
131	83	**ѓ**	163	A3	**J**	195	C3	**Г**	227	E3	**г**
132	84	**„**	164	A4	**¤**	196	C4	**Д**	228	E4	**д**
133	85	**…**	165	A5	**Ґ**	197	C5	**Е**	229	E5	**е**
134	86	**†**	166	A6	**¦**	198	C6	**Ж**	230	E6	**ж**
135	87	**‡**	167	A7	**§**	199	C7	**З**	231	E7	**з**
136	88	**€**	168	A8	**Ё**	200	C8	**И**	232	E8	**и**
137	89	**‰**	169	A9	**©**	201	C9	**Й**	233	E9	**й**
138	8A	**Љ**	170	AA	**Є**	202	CA	**К**	234	EA	**к**
139	8B	**‹**	171	AB	**«**	203	CB	**Л**	235	EB	**л**
140	8C	**Њ**	172	AC	**¬**	204	CC	**М**	236	EC	**м**
141	8D	**Ќ**	173	AD		205	CD	**Н**	237	ED	**н**
142	8E	**Ћ**	174	AE	**®**	206	CE	**О**	238	EE	**о**
143	8F	**Џ**	175	AF	**Ї**	207	CF	**П**	239	EF	**п**
144	90	**ђ**	176	B0	**°**	208	D0	**Р**	240	F0	**р**
145	91	**‘**	177	B1	**±**	209	D1	**С**	241	F1	**с**
146	92	**’**	178	B2	**І**	210	D2	**Т**	242	F2	**т**
147	93	**“**	179	B3	**і**	211	D3	**У**	243	F3	**у**
148	94	**”**	180	B4	**ґ**	212	D4	**Ф**	244	F4	**ф**
149	95	**•**	181	B5	**µ**	213	D5	**Х**	245	F5	**х**
150	96	**–**	182	B6	**¶**	214	D6	**Ц**	246	F6	**ц**
151	97	**—**	183	B7	**·**	215	D7	**Ч**	247	F7	**ч**
152	98		184	B8	**ё**	216	D8	**Ш**	248	F8	**ш**
153	99	**™**	185	B9	**№**	217	D9	**Щ**	249	F9	**щ**
154	9A	**љ**	186	BA	**є**	218	DA	**Ъ**	250	FA	**ъ**
155	9B	**›**	187	BB	**»**	219	DB	**Ы**	251	FB	**ы**
156	9C	**њ**	188	BC	**j**	220	DC	**Ь**	252	FC	**ь**
157	9D	**ќ**	189	BD	**S**	221	DD	**Э**	253	FD	**э**
158	9E	**ћ**	190	BE	**s**	222	DE	**Ю**	254	FE	**ю**
159	9F	**џ**	191	BF	**ї**	223	DF	**Я**	255	FF	**я**

Кодовая таблица KOI8-R (коды 128 – 255)

Код		Символ	Код		Символ	Код		Символ	Код		Символ
дес.	шестн.		дес.	шестн.		дес.	шестн.		дес.	шестн.	
128	80	─	160	A0	⌠	192	C0	ю	224	E0	Ю
129	81	│	161	A1	°	193	C1	а	225	E1	А
130	82	┌	162	A2	²	194	C2	б	226	E2	Б
131	83	┐	163	A3	ё	195	C3	ц	227	E3	Ц
132	84	└	164	A4	⌡	196	C4	д	228	E4	Д
133	85	┘	165	A5	²	197	C5	е	229	E5	Е
134	86	├	166	A6	┐	198	C6	ф	230	E6	Ф
135	87	┤	167	A7	┌	199	C7	г	231	E7	Г
136	88	┬	168	A8	┤	200	C8	х	232	E8	Х
137	89	┴	169	A9	┴	201	C9	и	233	E9	И
138	8A	┼	170	AA	┬	202	CA	й	234	EA	Й
139	8B	▀	171	AB	├	203	CB	к	235	EB	К
140	8C	▄	172	AC	─	204	CC	л	236	EC	Л
141	8D	█	173	AD	┼	205	CD	м	237	ED	М
142	8E	▌	174	AE	⌡	206	CE	н	238	EE	Н
143	8F	▐	175	AF	²	207	CF	о	239	EF	О
144	90	░	176	B0	²	208	D0	п	240	F0	П
145	91	▒	177	B1	²	209	D1	я	241	F1	Я
146	92	▓	178	B2	²	210	D2	р	242	F2	Р
147	93	⌠	179	B3	Ё	211	D3	с	243	F3	С
148	94	■	180	B4	²	212	D4	т	244	F4	Т
149	95	·	181	B5	²	213	D5	у	245	F5	У
150	96	√	182	B6	²	214	D6	ж	246	F6	Ж
151	97	≈	183	B7	²	215	D7	в	247	F7	В
152	98	≤	184	B8	²	216	D8	ь	248	F8	Ь
153	99	≥	185	B9	²	217	D9	ы	249	F9	Ы
154	9A		186	BA	²	218	DA	з	250	FA	З
155	9B	⌡	187	BB	²	219	DB	ш	251	FB	Ш
156	9C	°	188	BC	²	220	DC	э	252	FC	Э
157	9D	²	189	BD	²	221	DD	щ	253	FD	Щ
158	9E	·	190	BE	²	222	DE	ч	254	FE	Ч
159	9F	÷	191	BF	©	223	DF	ъ	255	FF	Ъ

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Дейтел Харви, М. Как программировать на C++ / Х.М. Дейтел, П.Дж. Дейтел; пер. с англ. под ред. А. Архангельского. М.: Бином, 2000. 1021 с.
2. Либерти Джесс Освой самостоятельно C++ за 21 день / Дж. Либерти, Б. Джонс. М.: Издательский дом «Вильямс», 2006. 784 с.
3. Программирование на C++ / А.Д. Хомоненко и др. М.: Альтекс, 2003. 512 с.
4. Кучин Н.В. Основы программирования на языке Си: учеб. пособие / Н.В. Кучин, М.М. Павлова. СПб.: СПбГУАП, 2001. 86 с.
5. Павловская Т.А. C/C++. Структурное программирование: практикум / Т.А. Павловская, Ю.А. Щупак. СПб.: Питер, 2003. 240 с.
6. Скляров В.А. Язык C++ и объектно-ориентированное программирование / В.А. Скляров. Минск: "Вышэйшая школа", 1997. 478 с.
7. Дэвис, Стефан, Р. C++ для «чайников» / С.Р. Дэвис. М.: Издательский дом «Вильямс», 2003. 336 с.
8. Либерти, Джесс. C++ Энциклопедия пользователя / Дж. Либерти. М.: ДиаСофт, 2001. 590 с.
9. Элджер, Джефф. C++ : библиотека программиста / Джефф Элджер. пер. с англ. Е. Матвеева. СПб.: Питер, 2000. 320 с.
10. Керниган Б. Язык программирования Си / Б. Керниган, Д. Ричи. пер. с англ.; под ред. В.С. Штаркмана. СПб.: Невский Диалект, 2001. 351 с.

11. Страуструп, Бьерн. Язык программирования C++. Специальное издание / Бьерн Страуструп; пер. с англ. С. Анисимова и М. Кононова; под ред. Ф. Андреева и А. Ушакова. М.: Бином-Пресс, 2007. 1098 с.
12. Страуструп, Бьерн. Дизайн и эволюция C++ / Бьерн Страуструп. СПб., М.: Питер: ДМК Пресс, 2006. 444 с.
13. Джосьютис, Николай. C++: Стандартная библиотека / Н. Джосьютис; пер.с англ. Е. Матвеева. СПб.: Питер, 2004. 727 с.
14. Богатырев, Андрей. Хрестоматия программирования на Си в UNIX / А. Богатырев // http://lib.ru/CTOTOR/book.txt
15. Джонсон, Майкл К. Разработка приложений в среде Linux / М.К. Джонсон, Э.В. Троан. М.: Издательский дом «Вильямс», 2007. 544 с.
16. Кубенский А.А. Структуры и алгоритмы обработки данных. Объектно-ориентированный подход и реализация на C++: учеб. пособие / А.А. Кубенский. СПб.: БХВ-Петербург, 2004. 464 с.
17. Бланшет Дж. QT4: программирование GUI на C++ / Дж. Бланшет, М. Саммерфилд. М.: Кудиц-Пресс, 2007. 736 с.
18. Шлее М. Qt4. Профессиональное программирование на C++ / М. Шлее. СПб.: БХВ-Петербург, 2007. 880 с.
19. Земсков Ю.В. Qt 4 на примерах / Ю.В. Земсков. СПб.: БХВ-Петербург, 2008. 608 с.

ОГЛАВЛЕНИЕ

Учебное издание

Корчагин Юрий Эдуардович

ПРОГРАММИРОВАНИЕ
НА ЯЗЫКАХ C и C++:
ЛАБОРАТОРНЫЙ ПРАКТИКУМ

В авторской редакции

Подписано в печать 07.07.2009.
Формат 60x84/16. Бумага для множительных аппаратов.
Усл. печ. л. 15,8. Уч.-изд. л. 14,5. Тираж 100 экз.
Заказ №_____

ГОУВПО «Воронежский государственный технический университет»
394026 Воронеж, Московский просп., 14